UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Desarrollo de sistema de control de velocidad para automóvil sincronizado con la base de datos de cinemómetros de la DGT

Autor/es

IVÁN PEÑA MORENO

Director/es

JAVIER RICO AZAGRA

Facultad

Escuela Técnica Superior de Ingeniería Industrial

Titulación

Grado en Ingeniería Electrónica Industrial y Automática

Departamento

INGENIERÍA ELÉCTRICA

Curso académico

2018-19

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL

TRABAJO DE FIN DE GRADO

TITULACIÓN: Grado en
Ingeniería Electrónica Industrial y Automática

CURSO: 2018/2019 **CONVOCATORIA: MARZO**

TÍTULO:

Desarrollo de sistema de control de velocidad para automóvil sincronizado con la base de datos de cinemómetros de la DGT

AUTOR: Iván Peña Moreno

DIRECTOR/ES: Javier Rico Azagra

DEPARTAMENTO: Ingeniería Eléctrica

IVÁN PEÑA MORENO

04-03-2019

RESUMEN/ABSTRACT

RESUMEN

El presente Trabajo Fin de Grado trata sobre el desarrollo de un sistema de control de la velocidad de un automóvil (Ford Focus TDCi DAW) mediante su aproximación con coordenadas GPS de los cinemómetros fijos, descargados de la base de datos de la Dirección General de Tráfico.

En un primer momento, se ha llevado a cabo un análisis de los componentes y señales necesarias: velocidad real del vehículo, posición de los pedales y señales del acelerador.

A continuación se prueba y configura un módulo GPS que sincronice la velocidad por proximidad a los radares. Un módulo Bluetooth para la visualización de los datos en la Radio-CD usando su mando para configurar las diferentes funciones.

Por último, una vez finalizados y probados todos los módulos, se procede a programar un microprocesador que controle automáticamente la velocidad, que sincronice la máxima permitida y limite la velocidad del vehículo, cuando se acerque a los cinemómetros.

ABSTRACT

The present End of Grade Project deals with the development of an automobile speed control system (Ford Focus TDCi DAW) using coordinates GPS of the recovered of the speed values of the database Traffic Administration.

In the first place, there has been carried out an analysis of the components and control signals: real speed of the vehicle, position of the pedals, signals of the throttle.

Next, it is tested and set a GPS module, that synchronizes the speed when the vehicle is nearby a radar. A Bluetooth module, is used to show the information by the Radio-CD device, using their pads to operate with the speed control.

At the end, when the modules are tested, the microprocessor is programmed in order to control the speed automatically, synchronizing the date of the max. speed limited and the speed of the vehicle, when the vehicle is nearby a radar.

UNIVERSIDAD DE LA RIOJA

ÍNDICE GENERAL

Desarrollo de Sistema de Control de Velocidad para automóvil sincronizado con la base de datos de cinemómetros de la DGT

UNIVERSIDAD DE LA RIOJA
GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Firma del presente documento:

Iván Peña Moreno

UNIVESIDAD DE LA RIOJA

Logroño, 3 de Marzo de 2019

MEMORIA

ANEXO I: ESQUEMAS ELÉCTRICOS FORD FOCUS

ANEXO II: CÓDIGO DE ARDUINO

PLANOS

PLIEGO DE CONDICIONES

PRESUPUESTO

UNIVERSIDAD DE LA RIOJA

MEMORIA

Desarrollo de Sistema de Control de Velocidad para automóvil sincronizado con la base de datos de cinemómetros de la DGT

UNIVERSIDAD DE LA RIOJA
GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Firma del presente documento:

Iván Peña Moreno

UNIVESIDAD DE LA RIOJA

Logroño, 3 de Marzo de 2019

MEMORIA

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE GRÁFICAS

ÍNDICE DE TABLAS

0. Introducción

Para la realización del TFG Desarrollo de sistema de control de velocidad para automóvil sincronizado con la base de datos de cinemómetros de la DGT, se ha utilizado un vehículo comercial del año 2002, un Ford FocusTDCi DAW.

Se ha llevado a cabo un análisis de los componentes y señales necesarias, como son la lectura de la velocidad real del vehículo, lectura de la posición de los pedales del vehículo, y lectura y clonación de la señal del acelerador. En su mayor parte la adquisición y manipulación de estas señales se logra de forma experimental por la falta de información.

Una vez logrado esto, se prueban y configuran distintos módulos externos para completar el proyecto, un módulo GPS, para la sincronización de velocidad por proximidad a los cinemómetros, un módulo Bluetooth, para la visualización de datos y modos de funcionamiento a través de un Smartphone y un aprovechamiento del mando del Radio-CD, el cual se convierte en el mando para el control de las distintas funciones.

Una vez finalizadas y probadas todas las partes por separado se procede a la programación de un microprocesador para el control de velocidad automático, la sincronización de velocidad por proximidad a los cinemómetros y una limitación de velocidad automática.

Ilustración1: Ford Focus TDCi Trend DAW

1. Objeto

El objetivo de este proyecto se trata de dotar a un vehículo sin asistencias electrónicas al control de velocidad con asistencias existentes de forma común en la mayoría de los vehículos fabricados en la actualidad y otras asistencias innovadoras como el control automático por proximidad a cinemómetros.

Todo ello con el uso de electrónica de bajo coste, como placas de desarrollo con microcontrolador Arduino y módulos electrónicos para las de posicionamiento GPS y comunicación Bluetooth. Para su realización, se han llevado a cabo las siguientes fases:

- Analizar los componentes del vehículo que pueden aportar la velocidad real y diseñar los circuitos electrónicos y programación para llevarlo a cabo.
- Analizar el acelerador del vehículo de forma experimental y diseñar los circuitos electrónicos y programación para llevar a cabo la manipulación electrónica del acelerador.
- Instalar y configurar el módulo GPS necesario para el cálculo de la proximidad de un cinemómetro.
- Instalar y programar el módulo Bluetooth, así como aplicación del Smartphone para la visualización de datos.
- Programar el microcontrolador en situación de funcionamiento real con todas las partes del proyecto y por último el conjunto de ellas.

2. Alcance

El alcance del proyecto es el control de velocidad automático sincronizadocon la proximidad a los cinemómetros o cualquier punto por coordenadas GPS que el usuario desee insertar en el programa, regulando el vehículo de forma autónoma la velocidad configurada para cada localización.

Aprovechando el diseño para la regulación automática por proximidad se dota al vehículo de distintos modos de regulación, como el control de crucero y el limitador de velocidad. Combinando las funciones se llevan a cabo seis modos de funcionamiento distintos:

- Modo 0: Controles automáticos desactivados.
- Modo 1: Limitador de velocidad por proximidad a radar fijo, detectado por coordenadas GPS.
- Modo 2: Limitador de velocidad por proximidad a radar fijo, detectado por coordenadas GPS y limitador de velocidad con asignación de consigna manual.
- Modo 3: Limitador de velocidad por proximidad a radar fijo, detectado por coordenadas GPS y control de velocidad con asignación de consigna manual o como se conoce de forma común, control de crucero.
- Modo 4: Control de velocidad con asignación de consigna manual o control de crucero.
- Modo 5: Limitador de velocidad con asignación de consigna manual.

También se ha realizado un amplio estudio de las posibilidades para la adquisición de señales y manipulación de las mismas:

- Adquisición e interpretación de las señales del indicador de velocidad del vehículo situado en el cuadro de instrumentos.
- Lectura y clonación de la señal del acelerador.
- Prueba y configuración del módulo externo GPS, para la sincronización de velocidad por proximidad a los cinemómetros.
- Prueba y configuración del módulo externo Bluetooth, para la visualización de datos y modos de funcionamiento a través de un Smartphone.
- Utilización del mando del Radio-CD, el cual se convierte en el mando para el control de las distintas funciones.

3. Antecedentes

En consecuencia de laevolución de la electrónica y su bajo coste, los vehículos modernos disponen de muchas asistencias a la conducción que la mayoría de la antigua flota automovilística carece, estas asistencias no estaban disponibles o suponían un coste elevado para ellas ofreciéndose como extras opcionales.

Este proyecto pretende demostrar que con los productos que existen en el mercado en la actualidad y sus reducidos costes, se puede dotar a un vehículo de prácticamente cualquier función de una forma segura y fiable. A continuación se detallan aquellos componentes electrónicos y comunicaciones en los que se basa la iniciativa de realizar este proyecto.

3.1 Hardware

3.1.1 Arduino UNO

El Arduino UNO es una placa de desarrollo con un microcontrolador popular por su código abierto, lo que da lugar a una gran cantidad de información disponible y a su vez un bajo coste. Esta placa incorpora un microcontrolador fabricado por ATMEL.

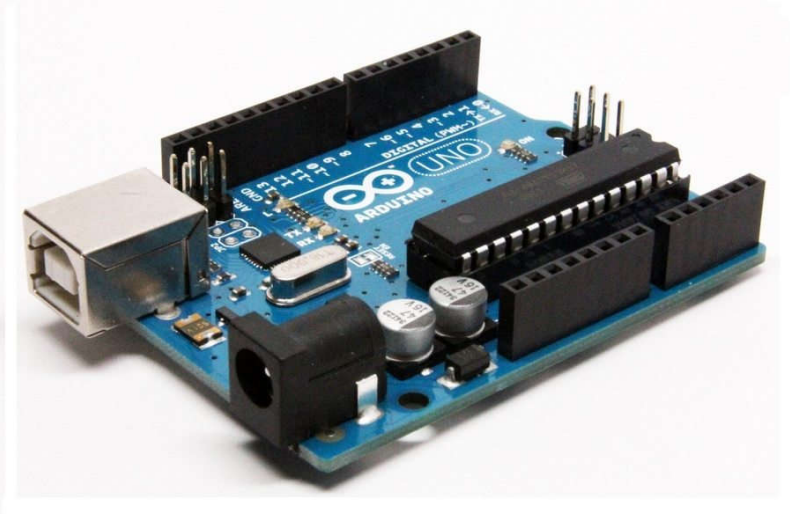


Ilustración 2: Placa de desarrollo Arduino UNO

Sus características son las siguientes:

- Voltaje: 5V
- Voltaje de entrada (recomendado): 7-12V

- Voltaje de entrada (límites): 6-20V
- Pines de entradas y salidas digitales: 14
- Pines de entrada analógica: 6
- Corriente en pines de entrada-salida: 40mA
- Corriente en pin de 3.3V: 50mA
- Memoria Flash: 32KB
- SRAM: 2KB
- EEPROM: 1KB
- Velocidad reloj: 16MHz
- Dimensiones: 68.6x53.4mm

Para el desarrollo de las diferentes partes del proyecto se ha utilizado esta placa por su sencillez y facilidad en la configuración, así como la gran cantidad de librerías que están a disposición.

Una de las ventajas será la posibilidad de comunicar con más de un puerto serie gracias a estas librerías, cosa que será de gran utilidad. Más adelante se decidirá si este u otro será el microcontrolador a usar, a medida de que el proyecto avance, ya que esto no es posible hasta terminar todo el proceso de investigación.

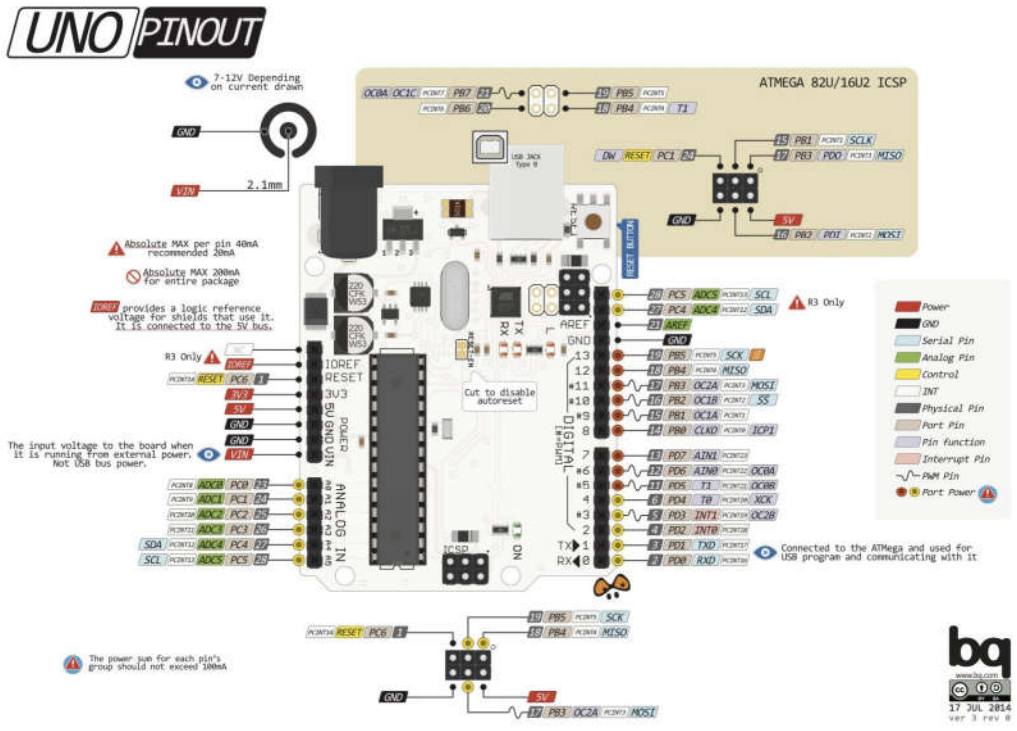


Ilustración 3: Asignación de pines en Arduino UNO

3.1.2 Interfaz CAN

El CAN-BUS es un bus de datos de uso industrial muy utilizado debido a su capacidad para transmitir datos a largas distancias, alta confiabilidad y velocidad media de transmisión. Es ampliamente utilizado en sistemas de diagnóstico vehicular, por lo que es una buena opción para la adquisición de datos en este proyecto y será la primera opción para la adquisición de datos.

La placa permite que el Arduino tenga la capacidad de operar en este tipo de redes.

En este proyecto se ha elegido una placapreparada para ello, que incluye los componentes necesarios:

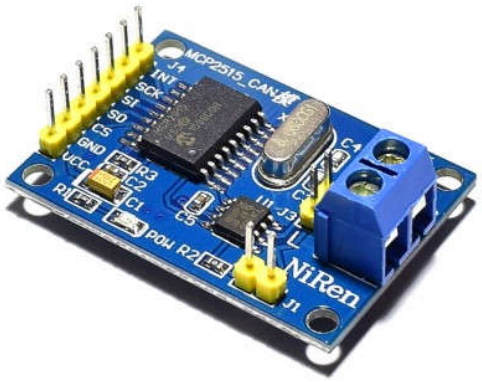


Ilustración 4: Interfaz CAN MCP 2515

Características del Interfaz CAN MCP 2515:

- MCP2515: Controlador autónomo CAN. Compatible con CAN 2.0A y CAN 2.0B, por lo que puede trabajar con tramas extendidas y con tramas estándar.
- Interface SPI de hasta 10 MHz.
- MCP2551: Es un transceiver CAN de alta velocidad (hasta 1MB/s) y hará de puente entre el controlador MCP2515 y el propio bus físico CAN. Implementa compatibilidad con el estándar ISO-11898.
- Data Frames standard (11 bit) y extendidos (29 bit).
- 2 buffers de recepción con priorización de mensaje.
- 2 LEDs indicadores.
- Cristal oscilador de 8 MHz: Es el encargado de sincronizar el controlador con el bus.

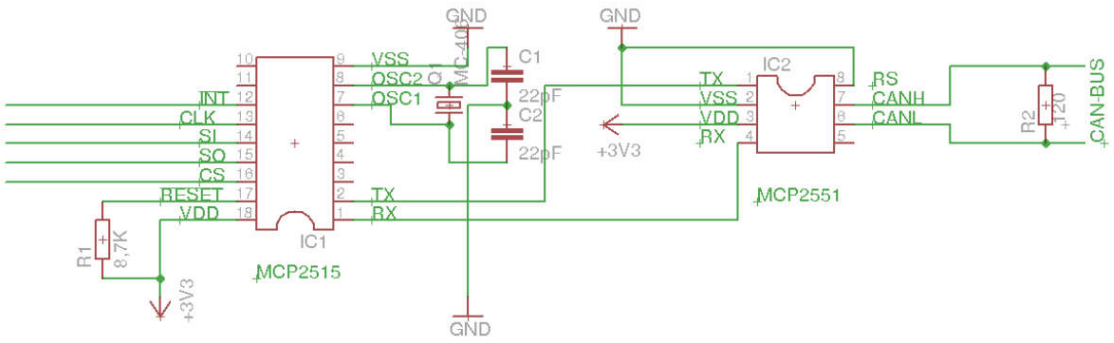


Ilustración 5: Circuito esquemático de interfaz CAN MCP 2515

3.1.3 Modulo GPS

Para la geolocalización es necesario otra placa específica ya que en la placa de desarrollo no tenemos esta funcionalidad. En este caso, se ha utilizado en módulo GPS con integrado NEO-6M, con una pequeña antena con baja precisión que se mantendrá en el proyecto ya que se busca la aproximación a unas coordenadas, no la precisión.

Ilustración 6: Módulo NEO-6M

Este módulo funciona alimentado con una tensión de alimentación de 3 a 5V y se comunica a través de Arduino con el puerto serie. La comunicación funciona por defecto a 9600bps y transmite la información a través del estándar NMEA.

Las tensiones de comunicación tendrán que ser reguladas ya que el microcontrolador Arduino dará tensiones de 5V superando los 3.3V de tensión de operación en el módulo, así como las tensiones de 3.3V de salida en el módulo si serán válidas ya que para el microcontrolador 3.3V es un nivel alto.

Además de los pines de conexión y comunicación, el módulo dispone de dos led indicadores de información, uno de alimentación correcta y el otro de señal de satélites. Un led indicará si envía las tramas de localización a través del puerto serie o si éste se queda fijo, indicará que no tenemos buena señal de GPS. Esto último es importante ya que aun estando en espacios abiertos puede llevar minutos en dar una localización.

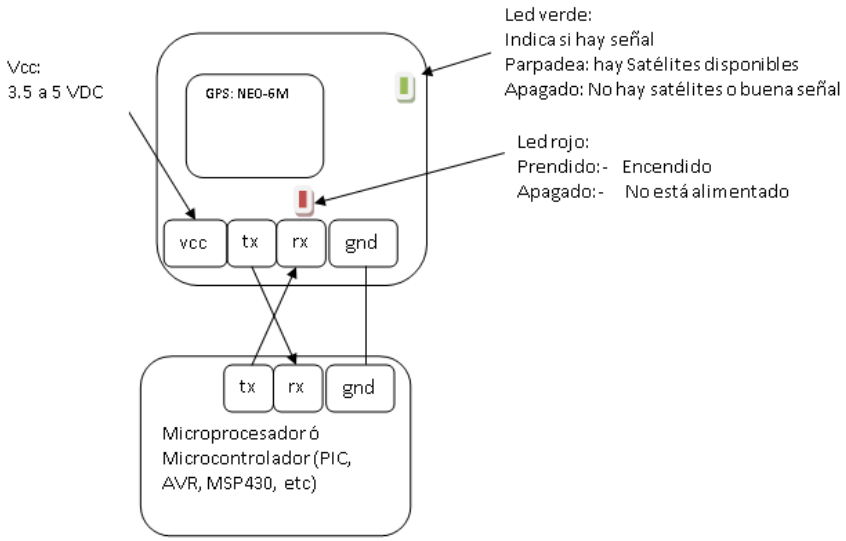


Ilustración 7: Funcionamiento de módulo NEO-6MAdemás este módulo tiene otras muchas características por defecto que podremos configurar a través del puerto serie.

Esquema del módulo de la placa en la que está el integrado NEO-6M:

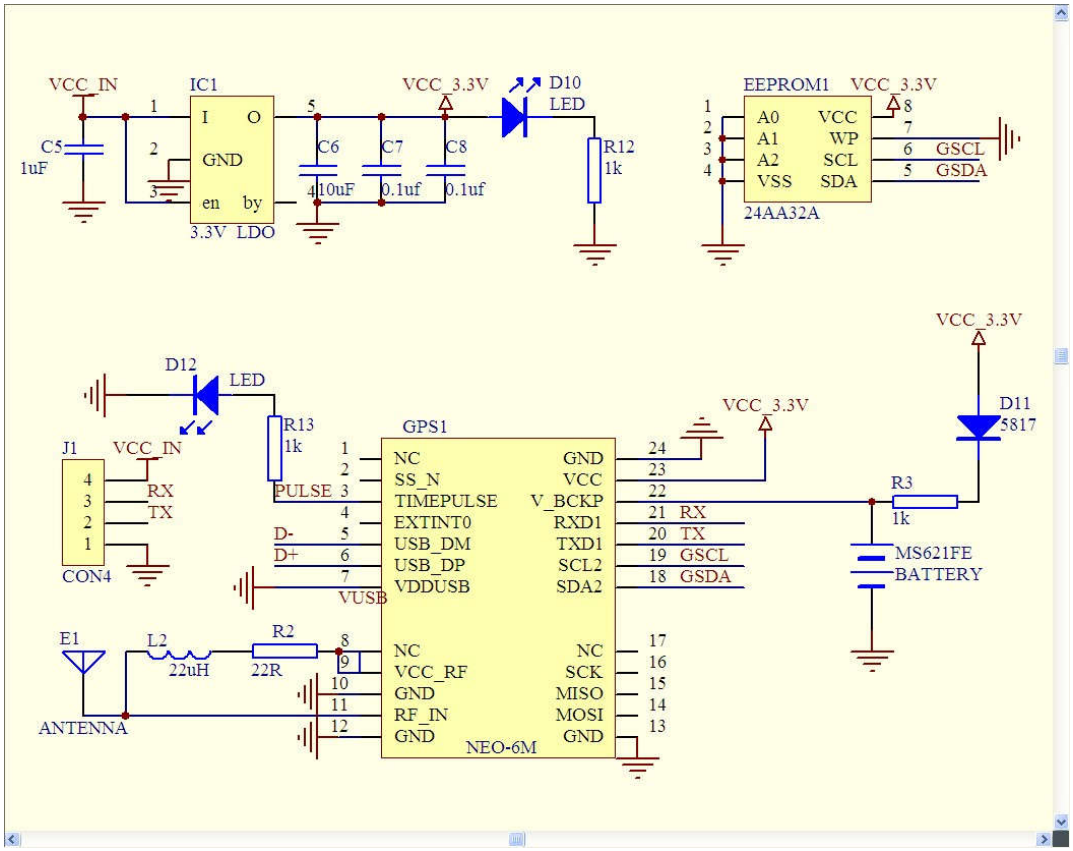


Ilustración 8: Circuito esquemático del módulo NEO-6M

Características:

- Modulo GPS Ublox NEO-6M
- Comunicación serial
- Voltaje de alimentación: (3.5 – 5)VDC
- Antena cerámica activa incluida
- LED indicador de señal
- Tamaño de antena 22x22mm
- Tamaño de módulo 23x30mm
- Batería incluida
- BAUDRATE: 9600
- EEPROM para guardar configuración de parámetros
- Sistema de coordenadas: WGS-84
- Sensibilidad de captura -148dBm
- Sensibilidad de rastreo: -161 dBm
- Máxima altura medible: 18000
- Máxima velocidad 515 m/s
- Exactitud: 1micro segundo
- Frecuencia receptora: L1 (1575.42 Mhz)
- Código C/A 1.023 Mhz
- Tiempo de inicio primera vez: 38s en promedio

- Tiempo de inicio : 35s en promedio

Configuración inicial:

- Velocidad de comunicación serial: 9800 baudios
- Pin de sincronización: desactivado

3.1.4 Módulo bluetooth

Para la visualización de datos en una App de Android que se diseña específicamente para este proyecto es necesaria la comunicación directa a un Smartphone. En este caso se elige un módulo Bluetooth entre el microprocesador y el Smartphone.

Los módulos Bluetooth se pueden configurar como maestro o esclavo, un maestro inicia el emparejamiento entre módulos para la conexión mientras que un esclavo espera a que se conecten otros a él.

En este caso se elige el módulo HC-06, exclusivamente de tipo esclavo, ya que el Smartphone será el encargado de iniciar la conexión y por lo tanto el emparejamiento.

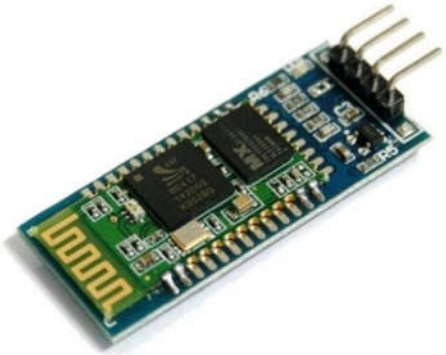


Ilustración 9: Módulo HC-06

El módulo que se ve en la ilustración lleva integrado un regulador de tensión admitiendo así tensiones desde los 3 a los 6V de alimentación y los pines respectivos para la comunicación serie con el microcontrolador, permitiendo así la comunicación entre el Smartphone y el Arduino.

Las tensiones de comunicación tendrán que ser reguladas ya que el microcontrolador Arduino dará tensiones de 5V superando los 3.3V de tensión de operación en el módulo Bluetooth, así como las tensiones de 3.3V de salida en el módulo si serán válidas ya que para el microcontrolador 3.3V es un nivel alto.

Características:

- Especificaciónbluetooth v2.0 + EDR (Enhanced Data Rate)
- Módulo montado en tarjeta con regulador de voltaje y 4 pines suministrando acceso a VCC, GND, TXD, y RXD
- Consumo de corriente: 30 mA a 40 mA
- Voltaje de alimentación: 3,3 V a 6 V
- Voltaje de operación: 3.3V
- Dimensiones totales: 1.7 cm x 4 cm aprox.
- Temperatura de operación: -25 ºC a +75 ºC
- Modo esclavo (Solo puede operar en este modo)
- Puede configurarse mediante comandos AT (Deben escribirse en mayúscula)
- Chip de radio: CSR BC417143
- Frecuencia: 2.4 GHz, banda ISM
- Modulación: GFSK (Gaussian Frequency Shift Keying)
- Antena de PCB incorporada
- Potencia de emisión: ≤ 6 dBm, Clase 2
- Alcance 5 m a 10 m
- Sensibilidad: ≤ -80 dBm a 0.1% BER
- Velocidad: Asincrónica: 2 Mbps (max.)/160 kbps, sincrónica: 1 Mbps/1 Mbps
- Seguridad: Autenticación y encriptación (Password por defecto: 1234)
- Perfiles: Puerto serial Bluetooth

Configuración inicial:

- Nombre del dispositivo Bluetooth: H06
- Velocidad de comunicación serial: 9800 baudios
- Contraseña de vinculación: 1234

3.1.5 Arduino MEGA

Arduino MEGA es otra placa de desarrollo al igual que el UNO, sigue siendo una placa de desarrollo de código libre y con microprocesador de ATMEL, en este caso el ATmega2560. Las funciones de la placa siguen siendo las mismas pero considerablemente mejoradas y ampliadas.

Ilustración 10: Placa de desarrollo Arduino MEGA

- Microcontrolador: ATmega2560
- Voltaje Operativo: 5V
- Voltaje de Entrada: 7-12V
- Voltaje de Entrada(límites): 6-20V
- Pines digitales de Entrada/Salida: 54 (de los cuales 15 proveen salida PWM)
- Pines analógicos de entrada: 16
- Corriente DC por cada Pin Entrada/Salida: 40 mA
- Corriente DC entregada en el Pin 3.3V: 50 mA
- Memoria Flash: 256 KB (8KB usados por el bootloader)
- SRAM: 8KB
- EEPROM: 4KB
- ClockSpeed: 16 MHz

El proyecto finalmente ha exigido aumentar las características del microprocesador, tanto en número de entradas y salidas como la cantidad de puertos serie para las comunicaciones.

Pines y conexiones:

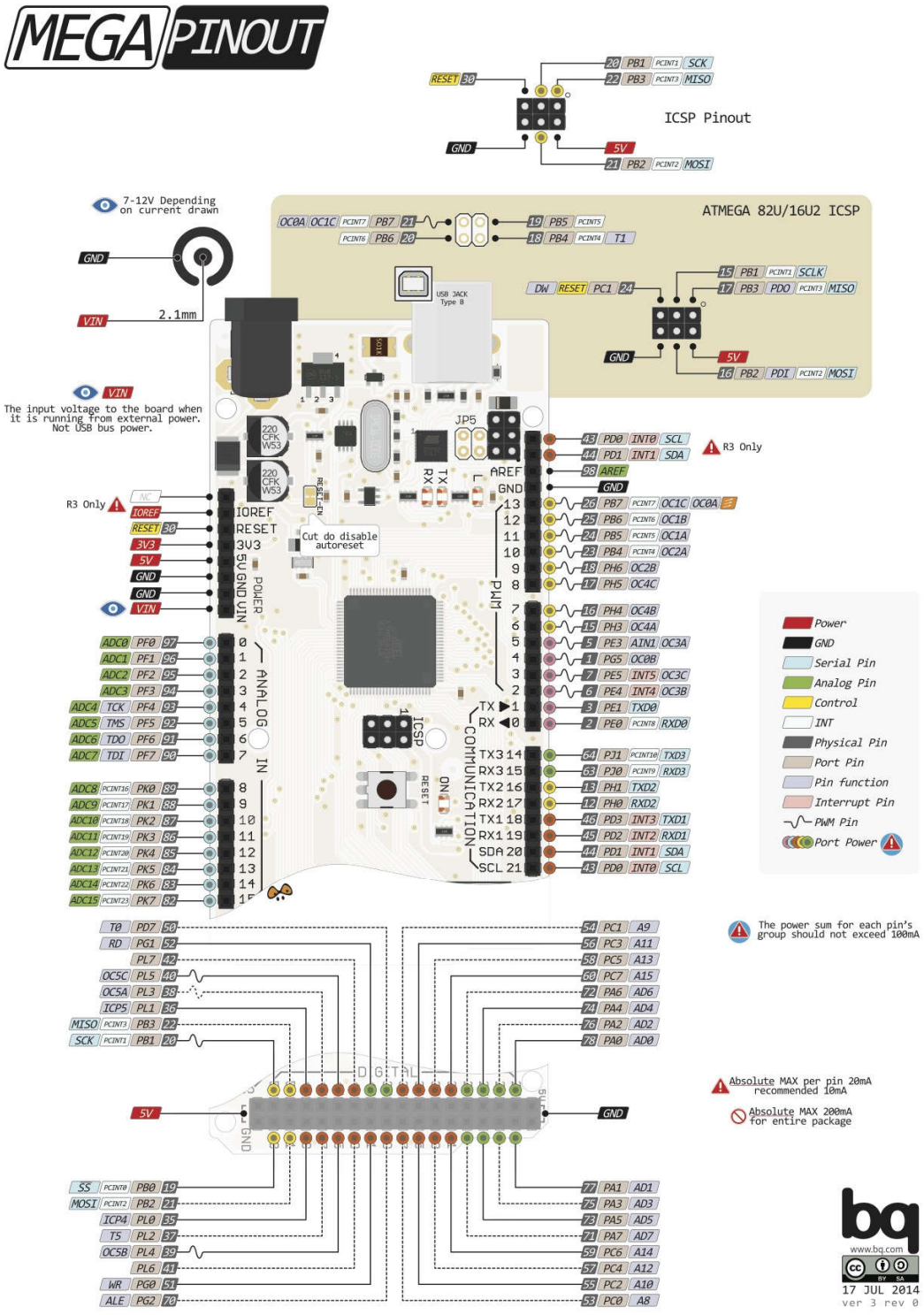


Ilustración 11: Asignación de pines de Arduino MEGA

3.1.5 Circuito integrado LM2917N

El LM2917N se trata de un integrado que da la posibilidad de conversión de pequeñas señales de sensores magnéticos a señales analógicas lineales amplificadas. Será de gran utilidad debido a que la gran mayoría de los vehículos disponen de este tipo de sensores.

Este integrado en su circuito interno incluye: un comparador de tensión en la entrada con una función de histéresis, una bomba de carga como convertidor frecuencia en tensión y un amplificador operacional con un transistor de salida.

El diagrama en bloques del circuito interno lo observamos en el siguiente diagrama:

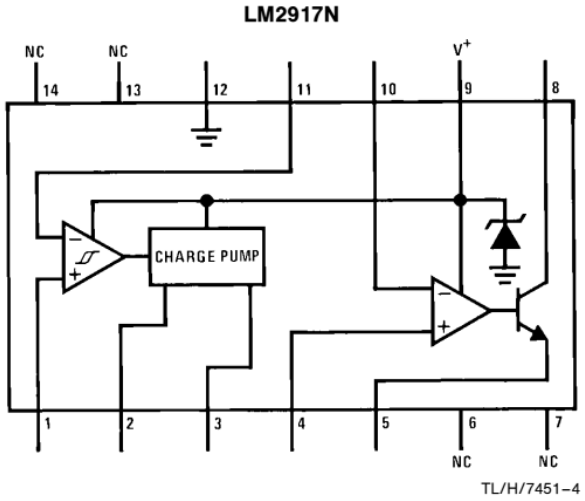


Ilustración 12: Diagrama de bloque de LM2917N

3.2 Comunicaciones

3.2.1 Conector OBD-II

Para la adquisición de datos para el control de velocidad, es imprescindible obtenerlos del vehículo en tiempo real, como la velocidad y otros datos que serán de utilidad si esto se hace posible.

Como norma, SAE J1939 establece que la ubicación del conector de diagnóstico debe estar ubicado en el habitáculo de ocupantes, debajo del panel de instrumentos, cercano al asiento del conductor o hasta 300 mm más allá de la consola central del vehículo. Todos estos requerimientos de ubicación deben cumplir con la facilidad de acceso y deben estar fuera de la línea visual de los

ocupantes. Estas características se cumplen, así como el tipo de conector y su cableado. Será en la realización del proyecto cuando se verá si esto es posible.

Standard	Pin 2	Pin 6	Pin 7	Pin 10	Pin 14	Pin 15
J1850 PWM	must have	-	-	must have	-	-
J1850 VPW	must have	-	-	-	-	-
ISO9141/14230	-	-	must have	-	-	optional
ISO15765 (CAN)	-	must have	-	-	must have	-

Ilustración 13: Asignación de pines para tipo de bus de comunicación.

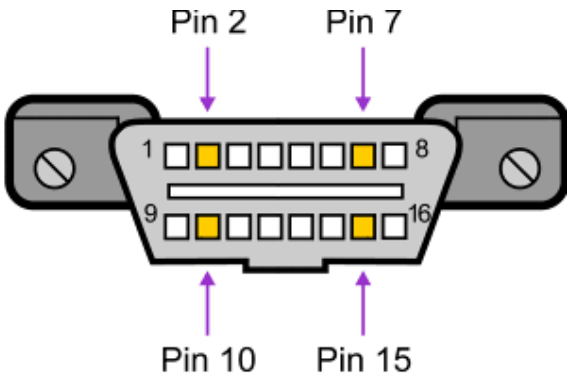


Ilustración 14: Vista frontal del conector OBD

En este proyecto se ha considerado la conexión por bus CAN, aunque existen otros tipos de comunicaciones en el conexionado del OBD-II, como los que se detallan en la siguiente tabla:

ISO9141-2	Older protocol used mostly on European vehicles between 2000 and 2004. Uses pins 7 and optionally 15.
SAE J1850 VPW	Diagnostic bus used mostly on GM vehicles. Uses pin 1, communication speed is 10.4 kB/sec.
SAE J1850 PWM	Diagnostic bus/protocol used mostly on Ford. Uses pins 1 and 2, communication signal is differential and it's rate is 41.6kB/sec.

3.2.2 Bus CAN

El bus CAN (ControllerArea Network), es un protocolo de comunicación desarrollado en 1983 por Bosch. Se utiliza en entornos distribuidos para comunicación de distintos nodos en una topología de bus. De esta manera se consigue eliminar cableado, reduciéndolo únicamente a los dos hilos del bus (CAN-H y CAN-L), y a los enganches.

En 1993 se estandarizó el bus CAN en la norma ISO 11898, en ella se recoge el estándar CAN 2.0A, que utiliza identificadores de dispositivo de 11 bits, y el estándar CAN 2.0B, que emplea un formato extendido de identificador de 29 bits. También se recoge el estándar CAN FD, que se está implementando en la actualidad, aunque en el presente proyecto no se va a utilizar.

En la norma también queda definida la máxima velocidad del bus, que será de 1Mbit/s para el bus CAN de alta velocidad, con resistencias finales de 120 Ω; y de 125 Kbit/s para el bus CAN de baja velocidad tolerante a fallos con resistencias de terminación de 100 Ω.

Los niveles lógicos se establecen en un par trenzado de hilos, que reducen el efecto del ruido y las interferencias. Estos dos hilos se denominan *CAN high*y*CAN low*, midiendo la diferencia de tensión entre ambos hilos se determina el estado del bus, este puede ser:

- Estado recesivo: Los dos hilos tienen el mismo nivel de tensión, según la norma ISO 11898, esta tensión estará comprendida entre -2 y 7 V.
- Estado dominante: En este estado existirá una diferencia de tensión en el bus de entre 1.5 y 3 V.

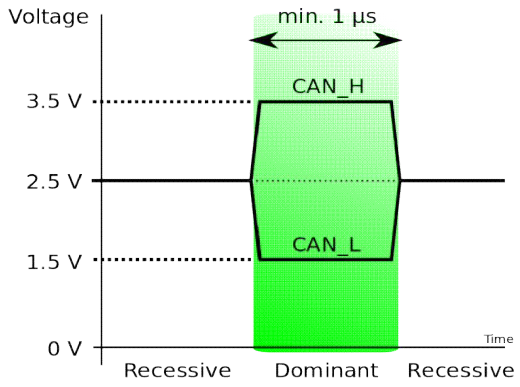


Ilustración 15: Niveles de tensión de las señales en el bus CAN

Trama de la comunicación bus CAN:

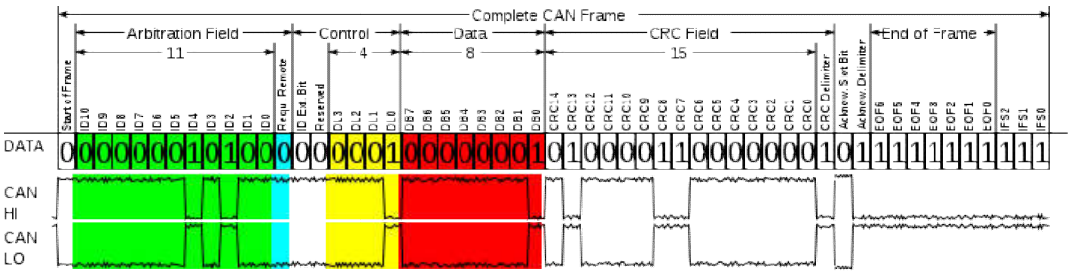


Ilustración 16: Trama de la comunicación bus CAN

3.2.3 Estándar NMEA para GPS

NMEA es el acrónimo de National Marine ElectronicsAssociation, esta agrupación fue formadamucho antes de que existieran los GPS para mejorar la comunicación entre aparatos de navegaciónelectrónicos. Actualmente es un estándar para los receptores GPS, que secomunican siguiendo las normas marcadas. Los mensajes se envíancodificados de manera quepueden ser interpretados por el microcontrolador que gestiona el modulo GPS:

Por ejemplo, el mensaje que interesa para el proyecto es "$GPRMC", este mensaje indica la siguiente información:

$GPRMC,155448.00,A,4227.50538,N,00228.85835,W,0.831,084.4,10219,003.1,W*6A

$GPRMC:

- 155448: Hora exacta en la que se da la información (15:54:48)
- A : Estado (A:Activo, V:Desactivado)
- 4227.50538,N: Latitud 42 grados, 27.50538' Norte
- 00228.85835,W: Longitud 11 grados, 8.85385' Oeste
- 0.831: Velocidad sobre el suelo en nudos
- 084.4: Angulo de seguimiento en grados
- 010219: Fecha actual (01/02/2019)
- 003.1,W: Variaciónmagnética
- *6a: Checksum

Existen muchos más mensajes con otro tipo de información, pero en este caso no seránecesario utilizarlos, ya que el mensaje "$GPRMC" da información suficiente para el propósito del presente trabajo.

4. Normas y referencias

4.1 Disposiciones legales y normativa aplicada

1) *Manual de Procedimientos de Inspección de las estaciones I.T.V versión 7.3.3.*

2) *Orden ITC/1620/2008, de 5 de junio, por la que se actualizan los anexos I y II del Real Decreto 2028/1986, de 6 junio, sobre las normas para la aplicación de determinadas directivas de la CE, relativas a la homologación de tipo de vehículos automóviles, remolques, semirremolques, motocicletas, ciclomotores y vehículos agrícolas, así como de partes y piezas de dichos vehículos.*

3) *Directiva 2007/46/CE del Parlamento europeo y del Consejo, de 5 de septiembre de 2007 por la que se crea un marco para la homologación de los vehículos de motor y de los remolques, sistemas componentes y unidades técnicas independientes destinadas a dichos vehículos (Directiva Marco).*

4) *El Real Decreto 736/1988, de 8 de julio, por el que se regula la tramitación de reformas de importancia de vehículos de carretera y se modifica el artículo 252 del Código de la Circulación.*

5) *El nuevo Real Decreto 866/2010, de 2 de julio, por el que se regula la tramitación de las reformas de vehículos*.

6) *Reglamento General de Vehículos Art. 5, "Vehículo prototipo o preserve de proyecto en fase de desarrollo, exento de homologación de tipo. Apto exclusivamente para matriculación a nombre del fabricante".*

7)*Identificación de acuerdo con el RD 866/2010, de 2 de julio, de la reforma a realizar y, en su caso, motivos que originan dicha realización y normativa aplicable en relación con los AR que puedan verse afectados por la reforma.*

8)*Real Decreto 2028/1986, de 6 de junio, por el que se dictan normas para la aplicación de determinadas Directivas de la CEE, relativas a la homologación de tipos de vehículos automóviles, remolques y semirremolques, así como partes y piezas de dichos vehículos.*

9)*Reglamento (UE) Nº 371/2010 de la Comisión de 16 de abril de 2010 que sustituye los anexos V, X, XV y XVI de la Directiva 2007/46/CE del Parlamento Europeo y del Consejo, por la que se crea un marco para la homologación de los vehículos de motor y de los remolques, sistemas, componentes y unidades técnicas independientes destinados a dichos vehículos.*

10) D*irectiva 70/157/CEE del consejo, de 6 de febrero de 1970, relativa a la aproximación de las legislaciones de los estados miembros sobre el nivel sonoro admisible y el dispositivo de escape de los vehículos a motor.*

11) *Reglamento (CE) nº 715/2007 del Parlamento Europeo y del Consejo, de 20 de junio de 2007, sobre la homologación de tipo de los vehículos de motor por lo que se refiere*

a las emisiones procedentes de turismos y vehículos comerciales ligeros (Euro 5 y Euro 6) y sobre el acceso a la información relativa a la reparación y el mantenimiento de los vehículos.

12)*Directiva 95/54/CE de la Comisión, de 31 de octubre de 1995, por la que se adapta al progreso técnico la Directiva 72/245/CEE del Consejo relativa a la aproximación de las legislaciones de los Estados Miembros sobre la supresión de parásitos radioeléctricos producidos por los motores.*

13)*Directiva 2005/21/CE de la Comisión, de 7 de marzo de 2005, por la que se adapta al progreso técnico la Directiva 72/306/CEE del Consejo, relativa a la aproximación de las legislaciones de los Estados miembros sobre las medidas que deben adoptarse contra las emisiones de contaminantes procedentes de los motores diésel destinados a la propulsión de vehículos.*

4.2 Bibliografía

1) *Salidas analógicas PWM en Arduino, Disponible en: https://www.luisllamas.es/salidas-analogicas-pwm-en-arduino/*

2) *Salidas Analógicas | Aprendiendo Arduino, Disponible en: https://aprendiendoarduino.wordpress.com/tag/salidas-analogicas/*

3) *Salida analógica mediante PWM y filtro paso bajo, Disponible en: https://www.luisllamas.es/salida-analogica-mediante-pwm-y-filtro-paso-bajo/*

4) *Salida analógica real con Arduino y DAC de 12bits MCP4725, Disponible en: https://www.luisllamas.es/salida-analogica-real-con-arduino-y-dac-de-12bits-mcp4725/*

5) *Tryit Editor v3.6, Disponible en: https://www.w3schools.com/html/tryit.asp?filename=tryhtml_formatting*

6) *Convertidores de frecuencia a voltaje, Disponible en: http://control-utb.blogspot.com/2010/09/convertidores-de-frecuencia-voltaje.html*

7) *Sistemas de Control Automático: septiembre 2010, Disponible en http://control-utb.blogspot.com/2010/09/convertidores-de-frecuencia-voltaje.html*

8) *SENSOR DE VELOCIDAD DEL VEHÍCULO (VSS), Disponible en: https://codigosdtc.com/sensor-vss/*

9) *Control RadioCD JVC Ford Focus//JVC-Ford Steering Wheel Control, Disponibleen: https://franprojects.wordpress.com/2010/09/22/jvc-steering-wheel-control/*

10) *Tutorial to Communicate Neo-6M GPS to Arduino, Disponibleen: https://www.instructables.com/id/How-to-Communicate-Neo-6M-GPS-to-Arduino/*

11) *Localización GPS con Arduino y los módulos GPS NEO-6, Disponible en: https://www.luisllamas.es/localizacion-gps-con-arduino-y-los-modulos-gps-neo-6/*

12) *Asignación de puertos Serial en Mega, Disponible en: https://forum.arduino.cc/index.php?topic=480455.0*

13) *Conecta un receptor GPS a tu Arduino - Estudio Roble, Disponible en: http://roble.uno/gps-arduino/*

14) *App inventor Arduino Sensor de humedad. Pantalla LCD, Bluetooth, Disponible en: http://kio4.com/appinventor/9M_humedad_BT.htm*

15) *Módulo Bluetooth HC-06 con Arduino y Android, Disponible en: http://arduparatodos.blogspot.com/2017/06/modulo-bluetooth-hc-06-con-arduino-y.html*

16) *Reliable Bluetooth CommsBetween Arduino and MIT App Inventor (AI2), Disponibleen:https://arduinoplusplus.wordpress.com/2017/05/10/reliable-bluetooth-data-transfer-between-arduino-and-mit-app-inventor-ai2/*

17) *TFG "Lenguaje visual para la especificación de sistemas Teleo-Reactivos utilizando herramientas tipo Scratch" Pablo Rubio Ibáñez*

18) *Bluetooth HC-06 | DYOR: Do Your Own Robot, Disponibleen: http://dyor.roboticafacil.es/bluetooth-hc-06/*

19) *Control RadioCD JVC Ford Focus//JVC-Ford Steering Wheel Control, Disponibleen: https://franprojects.wordpress.com/2010/09/22/jvc-steering-wheel-control/*

20) *Mit app inventor 2 – Inventor de aplicaciones, Disponible en: https://inventordeaplicaciones.es/tag/mit-app-inventor-2/*

21) *App inventor Servicios, Disponible en: http://kio4.com/appinventor/35movilenpausa.htm*

22) *Josemaco.files.wordpress.com, Diagrama Cableado Focus, Disponible en: https://josemaco.files.wordpress.com/2011/03/focus-dc-12-20051.pdf*

23) *Ford OBD-II diagnostic interface pinoutdiagram, Disponible en: http://pinoutguide.com/CarElectronics/ford_obd_2_pinout.shtml*

24) *Información sobre tramos y puntos de intensificación de vigilancia de la velocidad, Disponible en: http://www.dgt.es/es/el-trafico/control-de-velocidad/index-radares.shtml*

25) *Calcular distancia a partir de datos GPS, Disponible en: http://www.deif.org/blog/calcular-distancia-a-partir-de-datos-gps/*

26) *Radares fijos. Dgt.es, Disponible en: http://www.dgt.es/images/Tramos_INVIVE_17102017.xml*

27) *OBD2 protocols, Disponible en: http://www.obdtester.com/obd2_protocols*

28) *Arduino uno Data, Disponible en: https://docs-emea.rs-online.com/webdocs/0e8b/0900766b80e8ba21.pdf*

29) *CAN-BUS Shield V1.2 - Seeed Wiki, Disponibleen: http://wiki.seeedstudio.com/CAN-BUS_Shield_V1.2/*

30) *Biblioteca.unirioja.es. "Gestión telemática de vehículos agrícolas." Alberto Úbeda Cañas. Disponible en: https://biblioteca.unirioja.es/tfe_e/TFE002820.pdf*

31) *Módulo GPS Ublox NEO-6M v2 con memoria EEPROM, Disponible en: https://electronilab.co/tienda/modulo-gps-ublox-neo-6m-v2-con-memoria-eeprom/*

32) *App inventor Arduino Sensor de humedad. Pantalla LCD, Bluetooth. Disponible en: http://kio4.com/appinventor/9M_humedad_BT.htm*

33) *Módulo Bluetooth HC-06 con Arduino y Android Disponible en: http://arduparatodos.blogspot.com/2017/06/modulo-bluetooth-hc-06-con-arduino-y.html*

34) *Reliable Bluetooth CommsBetweenArduino and MIT App Inventor (AI2), Disponible en:https://arduinoplusplus.wordpress.com/2017/05/10/reliable-bluetooth-data-transfer-between-arduino-and-mit-app-inventor-ai2/*

35) *Módulo Bluetooth HC-06 Serial Rs232 (TTL) – Electronilab, Disponible en: https://electronilab.co/tienda/modulo-bluetooth-hc-06-serial-rs232ttl/*

36) *Arduino Mega: Características y Capacidades | PanamaHitek, Disponible en: http://panamahitek.com/arduino-mega-caracteristicas-capacidades-y-donde-conseguirlo-en-panama/*

37) *ArduinoBoards-Pin mapping–iCircuit, Disponibleen: http://icircuit.net/arduino-boards-pin-mapping/141*

38) *Diagrama de pines Arduino (pinoutarduino) - Geek Factory, Disponible en:https://www.geekfactory.mx/tutoriales/tutoriales-arduino/diagrama-de-pines-arduino-pinout-arduino/*

5. Definiciones y abreviaturas

5.1 Definiciones

- Arduino: Compañía de hardware libre que desarrolla placas basadas en microcontrolador de bajo coste. También produce el entorno de desarrollo para estas placas.
- Interfaz: Permite el intercambio de información entre dos sistemas diferentes, o entre un sistema y una persona.
- Adquisición de datos: Proceso por el cual se mide un fenómeno eléctrico o físico y a partir de él se obtienen datos experimentales que proporcionan una información veraz.
- Máquina de estados: Estructura de programación que definirá el comportamiento de un sistema en función del estado en que este se encuentra. Cada estado define las condiciones de funcionamiento del sistema a partir de una serie de variables.
- Prototipo: Modelo o primera aproximación de un invento, a partir del cual se puede crear un producto final.
- Hardware: Elementos físicos que constituyen un sistema electrónico o informático.
- Software: Conjunto de programas que definen el comportamiento de los elementos físicos de un sistema electrónico o informático.

5.2 Abreviaturas

- OBD: OnBoardDiagnostics (Diagnóstico de a bordo). Se trata de un conector que verifica todos los sensores y actuadores del vehículo, desde los años 80 ha ayudado a los talleres en el diagnóstico de averías.
- ABS: Anti-lock Braking System. Se trata del sistema antibloqueo de frenos para la mejora de frenada y la dirección en frenadas bruscas o pavimentos deslizantes.
- SPI: Serial Peripheral Interface. Estándar de transmisión de datos entre circuitos integrados en sistemas electrónicos. Trabaja de manera síncrona. Permite el multiplexado de la activación de dispositivos para comunicar con varios dispositivos simultáneamente.
- GPS: Global PositioningSystem. Sistema desarrollado por el departamento de defensade los EEUU, permite el posicionamiento de un objeto en cualquier punto de la tierra mediante triangulación con ayuda de 24 satélites.

- NMEA: National Marine Electronics Association. Organización estadounidense queestablece los estándares de la electrónica marina. Desarrolló el estándar por el que secomunican los receptores GPS.
- CAN: ControllerArea Network. Protocolo de comunicaciones basado en una topologíade bus. Muy extendido en la industria del automóvil y en todo tipo de vehículos especiales.

6. Requisitos de diseño

Los requisitos del diseño para dotar al vehículo de asistencias electrónicas al control de velocidad y otras asistencias innovadoras como el control automático por proximidad a cinemómetros, se han adecuado a las necesidades en los análisis y en la adquisición de datos.

Todo ello se ha realizado con el uso de electrónica de bajo coste, como placas de desarrollo con microcontrolador Arduino y módulos electrónicos para las de posicionamiento GPS y comunicación Bluetooth.

El conjunto de todas las funciones que se detallan a continuación son las que indicaran las características mínimas con las que se diseñara el proyecto; como el número de entradas analógicas necesarias, tipos de comunicación, módulos externos y circuitos electrónicos auxiliares.

- Requisitos para la lectura de la velocidad real y del diseño de los circuitos electrónicos.
- Manipulación electrónica del acelerador.
- Adquisición de coordenadas GPS para el cálculo de la proximidad de un cinemómetro.
- Comunicación Bluetooth con el Smartphone para la visualización de datos.
- Manipulación y configuración del control, por parte del usuario.
- Elección de un microprocesador para el conjunto de las funciones.

6.1 Control de aceleración.

Para el control de crucero así como el control de velocidad en proximidad a un radar fijo mediante coordenadas GPS es necesario el control de consigna de aceleración. Esto se ha conseguido a través de la lectura de las señales del pedal del aceleración y clonación de las mismas o asignando valores dependiendo si lo que se quiere es el funcionamiento normal o el control de la aceleración.

En primer lugar se realiza una búsqueda de esquemas del vehículo del proyecto, un Ford Focus 1.8 TDCI DAW. Se encuentran dos partes significativas que son el esquema general y la conexión a la unidad de control central (centralita) y el conector del acelerador con su respectiva numeración para los pines. Con esta información se asignan tres pines analógicos de entrada para la lectura de la posición del pedal del acelerador y tres pines de salida PWM para el control de velocidad o clonación de las señales de entrada dependiendo de las necesidades en cada momento.

Ilustración 17: Esquema reducido del pedal del acelerador

6.2 Adquisición de señales para la lectura de velocidad

Se han desarrollado cuatro formas diferentes para realizar la lectura de la velocidad real del vehículo, aunque solo una ha sido efectiva.

Todas ellas podrían ser válidas en otros proyectos, con otro tipo de vehículo o con otro tipo de componentes.

Es importante comentar que existe una quinta forma: la toma de la velocidad a través del dispositivo GPS del proyecto. Ésta, se ha tenido en cuenta y desestimado por la poca fiabilidad en el tiempo de refresco de la misma y la inseguridad de tener siempre disponible esa señal gps.

Como se detalla en el apartado *Análisis de las soluciones*, se realiza la toma de datos de velocidad directamente desde el indicador del cuadro de instrumentos, realizando una lectura de las tensiones en las cuatro conexiones de las dos bobinas que tiene el indicador. Esto exige añadir cuatro entradas analógicas para la lectura de los cuatro valores y el diseño de cuatro divisores de tensión para no superar los valores máximos del microcontrolador Arduino.

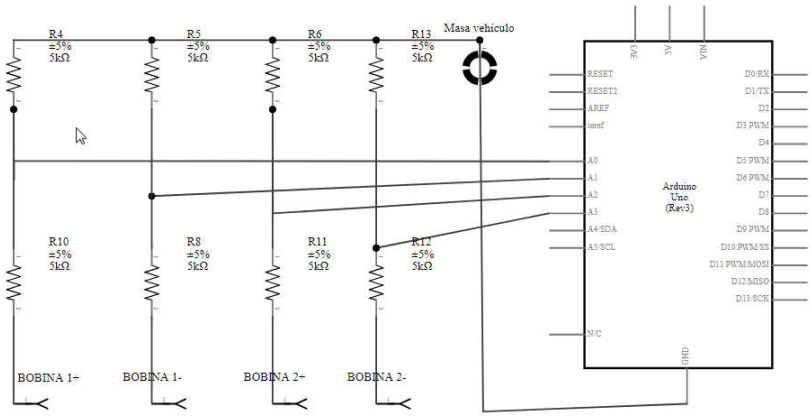


Ilustración 18: Circuito esquemático de la conexión al indicador de velocidad

6.3 Adquisición de coordenadas GPS

Para la geolocalización es necesaria otra placa específica, ya que en la placa de desarrollo de Arduino no tenemos esta funcionalidad. En este caso se ha utilizado el módulo GPS con integrado NEO-6M y comunicación Serie, por lo que se hace necesario un puerto adicional de comunicación en el microcontrolador.

Ilustración 19: Módulo NEO-6M

6.4 Comunicación Bluetooth con el Smartphone

Para la visualización de datos en tiempo real se diseña una App de Android específicamente para el proyecto. En este caso se elige un módulo Bluetooth para dicha conectividad entre el microprocesador y el Smartphone.

Para la visualización de los datos en el Smartphone es necesario el diseño de la aplicación, que en este proyecto se lleva a cabo a través de un software libre en plataforma Web, App Inventor. También es necesario asignar otro puerto de comunicación serie con el microcontrolador.

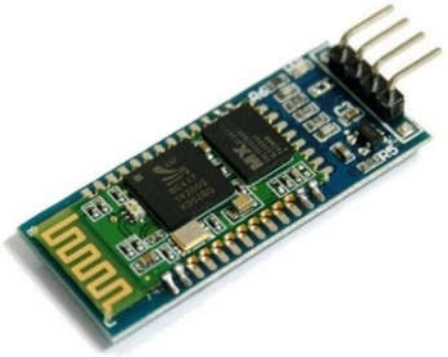
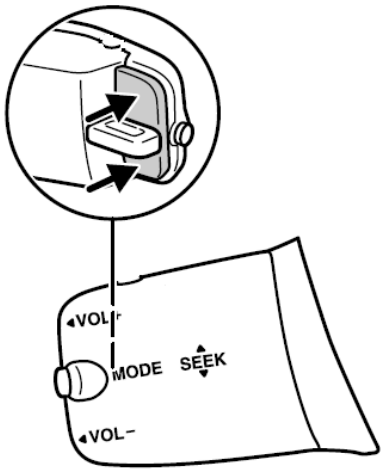


Ilustración 20: Módulo HC-06

6.5 Manipulación y configuración del control

Para el control de velocidad, así como la activación y desactivación de funciones se ha aprovechado el mando del Radio-CD que existía en el equipamiento de serie del vehículo. Se trata de un mando con cinco botones, más de los necesarios, así que es la opción más interesante, sobre todo por su posición al alcance de las manos en el volante, como si de un mando de control de crucero original se tratase.

Para la adaptación del mando a las necesidades del proyecto será necesario un circuito de alimentación y una adaptación de la señal de salida. Para la lectura y selección del botón pulsado, se asigna una entrada analógica en el microcontrolador para la lectura de la señal del mando.



6.6 Requisitos del microprocesador

Para cumplir con todas las necesidades del proyecto es necesario aumentar las posibilidades del Arduino UNO.

Arduino MEGA es otra placa de desarrollo al igual que el UNO, sigue siendo una placa de desarrollo de código libre y con microprocesador de ATMEL, en este caso el ATmega2560. Las funciones de la placa siguen siendo las mismas pero considerablemente mejoradas y ampliadas.

La placa de desarrollo Mega cumple con las necesidades para las conexiones requeridas:

- 8 entradas analógicas: velocímetro, acelerador y mando Radio-CD.
- 3 salidas PWM para el control del acelerador.
- 3 puertos de comunicación serie: módulo GPS, módulo Bluetooth y modo monitor para la visualización de datos a través de PC.

7. Análisis de las soluciones

7.1 Control de aceleración

Para el control de crucero así como el control de velocidad en proximidad a un radar fijo mediante coordenadas GPS es necesario el control de consigna de aceleración. Esto se ha conseguido a través de la lectura de las señales del pedal del aceleración y clonación de las mismas o asignando valores dependiendo si lo que se quiere es el funcionamiento normal o control de la aceleración.

En primer lugar se realiza una búsqueda de esquemas del vehículo del proyecto, un Ford Focus 1.8 TDCI DAW. Se encuentran dos partes significativas que son el esquema general y conexión a la unidad de control central (centralita) y el conector del acelerador con su respectiva numeración para los pines.

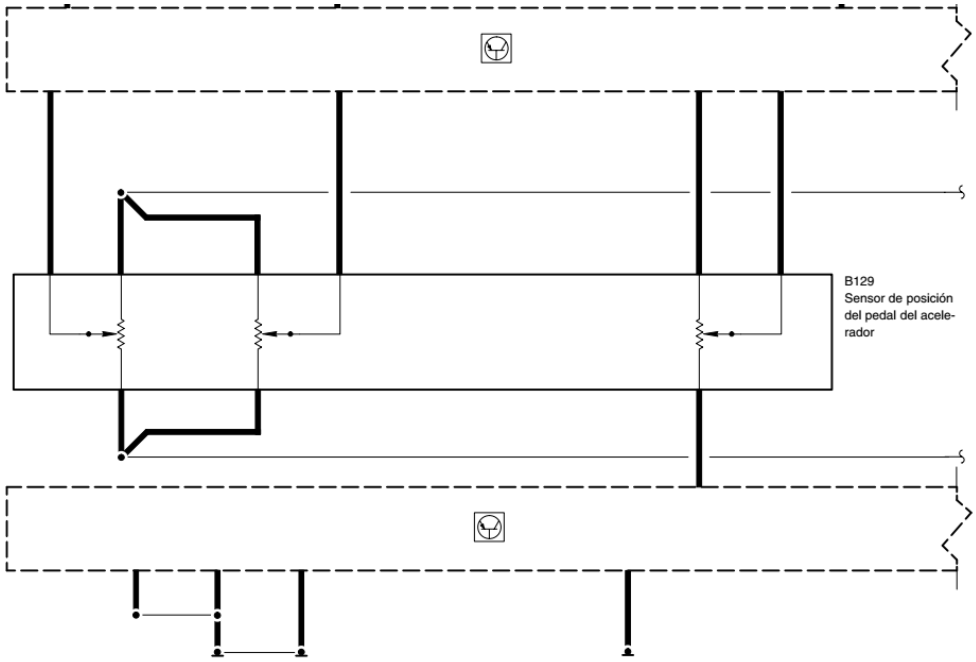


Ilustración 21: Esquema reducido del pedal del acelerador

Hasta aquí se puede observar que se debe clonar al menos 3 señales para “engañar” al módulo del control del motor del vehículo, pero no se sabe qué tipo de señal son, el rango de las mismas, ni qué tensiones de alimentación se tienen en las alimentaciones de los potenciómetros.

En el siguiente esquema se observan las conexiones de una forma más detallada aunque sigue faltando información, por lo tanto se tomaran medidas desmontando y analizando el sensor de acelerador.

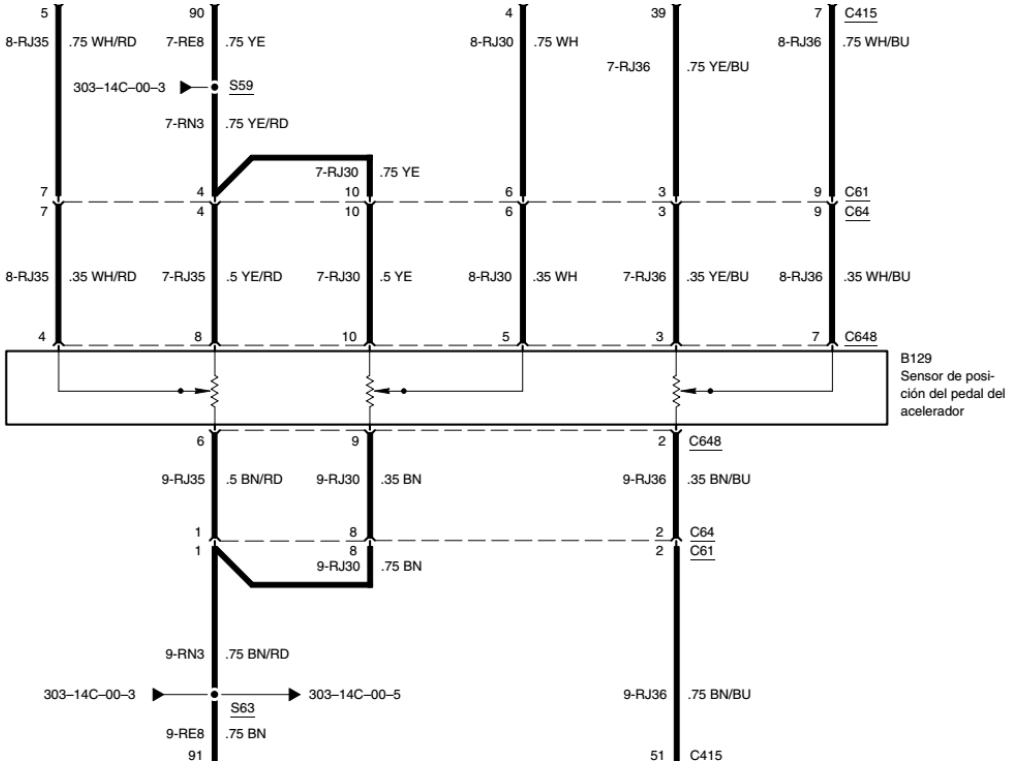


Ilustración 22: Conexiones al pedal del acelerador

Analizando el sensor de posición del pedal del acelerador se observa que los esquemas del vehículo no reflejan de forma fiel el hardware del acelerador, midiendo las resistencias en los terminales 4, 5 y 7 se observa que hay otras resistencias en serie en su interior y que una de las salidas dará tensiones con pendiente inversa a las otras dos, siendo esta una tabla de los datos tomados en los limites mínimo y máximo del rango del acelerador.

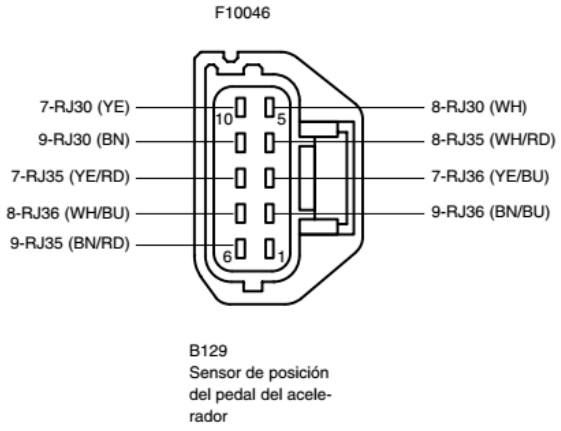


Ilustración 23: Conector y descripción de pines en el acelerador

Pin	Acelerador 0% (Ω)	Acelerador 100% (Ω)
R_6-4	2,2K	430
R_9-5	650	2K
R_2-7	460	2,2K
R_6-8	2,54K	2,54K
R_9-10	2,54K	2,54K
R_2-3	3,3K	3,3K

Tabla 1: Valores en Ω en el acelerador

Con la toma de datos de forma experimental se confirma que no tiene mucho que ver el esquema del manual de taller con la realidad, pero de esta forma ya se sabe o intuye qué disposición de conexionado existe en el interior del acelerador.

Una vez conocido esto, se toman datos reales de tensiones a las salidas y entradas de los pines de alimentación del pedal del acelerador, ya que estos datos van a ser más significativos a la hora de clonar el acelerador, ya que serán esas tensiones las que se deben clonar con el microcontrolador. Todas estas tensiones se miden respecto a masa, esto es muy importante ya que se puede ver en los esquemas que tanto la entrada como la salida del potenciómetro viene dada desde el módulo de control, lo que podría dar lugar a señales diferenciales más complejas de clonar.

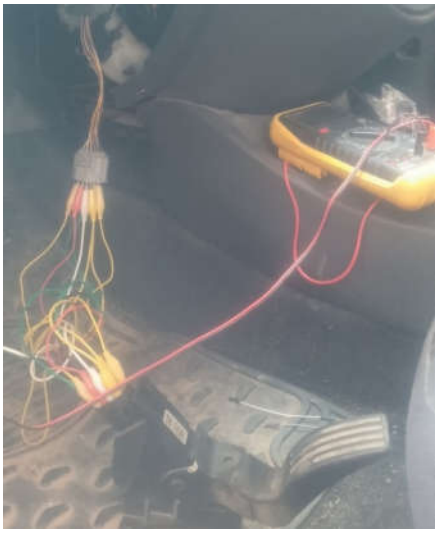


Ilustración 24: Conexión con bananas para la lectura de las salidas

Pin	Acelerador 0% (Vdc)	Acelerador 100% (Vdc)
V4	1,29	3,9
V5	4,33	0,84
V7	0,7	3,33
V3	5	5
V8	4,9	4,9
V10	4,9	4,9
V6	0,1	0,1

V9	0,1	0,1
V2	0	0

Tabla 2: Valores de tensión en las conexiones del acelerador, respecto a GND del automóvil

Una vez se ha realizado la tabla con los límites de las tensiones tanto con el pedal sin pisar y pisado al 100%, se observa las tensiones fácilmente simuladas por microcontrolador, sin necesidad de más electrónica adicional. Es de gran importancia medir estos valores desde la masa del vehículo, ya que se observa cómo no está conectado directamente a masa uno de los terminales de los potenciómetros en paralelo, dando 0.1 voltios, así como su alimentación está por debajo del tercer potenciómetro con 4.9 voltios.

Haciendo unas medidas se observa una resistencia de 50Ω para las masas que proporciona la unidad de control a alguno de los sensores, esto se puede deber a tener controlado en todo momento la conexión segura de los sensores relacionados con la seguridad del vehículo. Será cuando se clonen las señales en el momento que se verá si la unidad de control detecta algo inusual o no.

7.1.1 Prueba de control de aceleración

Con el análisis de la adquisición de datos anterior se procede a una prueba de control de aceleración real, es decir, se simulará una consigna con un potenciómetro accionando así el acelerador.

Con ayuda de una protoboard y el microcontrolador Arduino se generan las señales de los pines de salida del acelerador que serán el reflejo de una entrada analógica controlada con un potenciómetro externo, esto simulará el acelerador.

Como ya se ha descrito en apartados anteriores la placa de Arduino no dispone de señales analógicas de salida por lo que se hace necesaria una conversión de la señal modulada PWM a señal analógica, para ello, existen varias opciones, como por ejemplo:

- Conversor digital analógico. El MCP4725 es un conversor controlado a través de I2C de sencilla configuración y control a través del microcontrolador, precisión de 12 bits y muy económico.

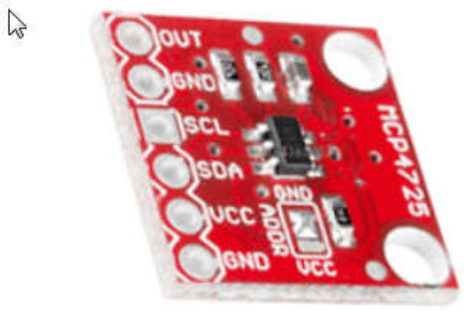


Ilustración 25: Conversor MCP4725

- Circuito de paso bajo RC.

En este caso se usa un circuito RC para cada una de las salidas, es algo básico y sencillo pero suficiente.

Aclarar que se ha probado la misma configuración sin el filtrado RC y la prueba ha sido igual de satisfactoria, lo que da lugar a intuir que ya existe un filtrado de la señal en la unidad de control y por lo tanto, con la señal filtrada se puede considerar muy válida.

Cálculo del filtro paso bajo para las salida PWM

Se trata de suavizar la señal para que se parezca al ejemplo, no se va conseguir dejar una señal totalmente plana, pero si dependerá la calidad de la misma cuanto más suave se quede.

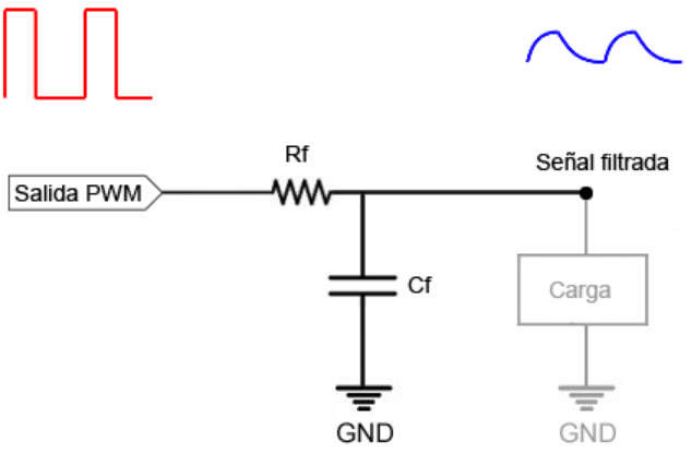


Ilustración 26: Esquema filtro RC

Existen dos parámetros para ajustar el filtro RC:

- El tiempode respuesta: tiempo que tardará en establecerse el valor deseado.

- Rizado: es la oscilación que difiere del calor medio.

Como se observa en el ejemplo, mejorar uno de los parámetros dará lugar a empeorar el otro, esto siempre que se esté hablando de no variar la frecuencia.

Variar la frecuencia de la señal PWM es posible aunque en este caso no será necesario.

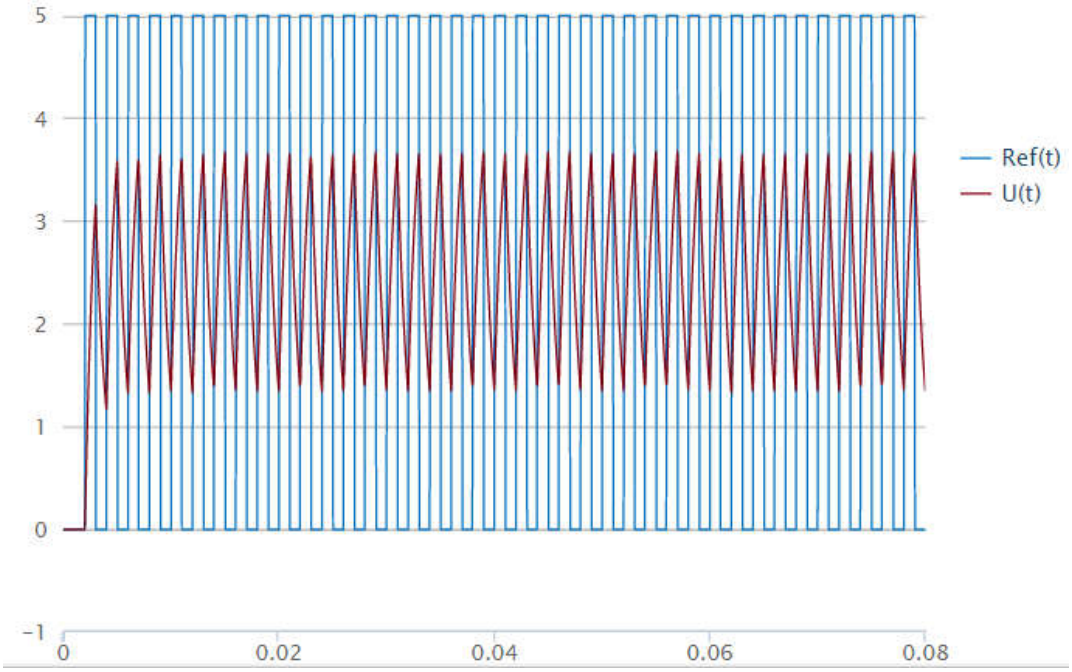


Ilustración 27: Ejemplo de salida del filtro RC

Simulando un escalón de 0 a 2,5V se obtiene la configuración elegida para los siguientes datos de entrada:

- Escalón: 2,5V
- Frecuencia de PWM: 500Hz
- Valor de señal PWM: 0-5V
- Constante de tiempo (tau): 0.004s
- Tiempo de establecimiento para el 90%: 0.0921
- Resistencia: 40kΩ
- Condensador: 1uF

Esta última configuración es elegida principalmente por el establecimiento para el 90%, este valor es límite de establecimiento, ya que el software de control trabajará a 0.1 segundos de ciclo.

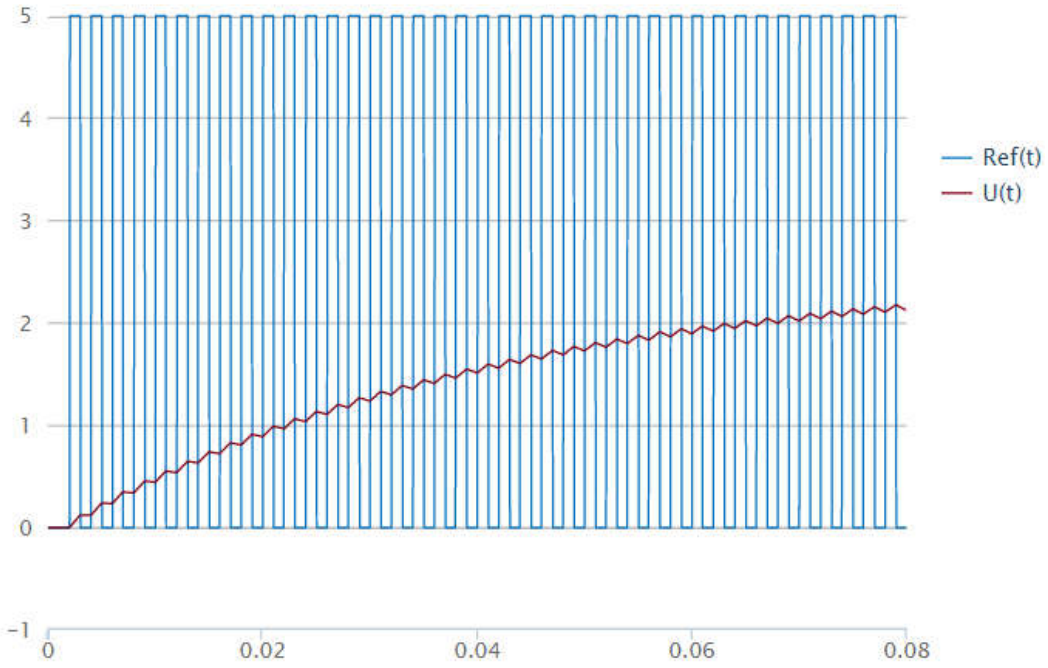


Ilustración 28: Simulación de salida RC elegida

7.1.2 Conexión y prueba del circuito del acelerador

Una vez calculados los componentes que se van a usar se realiza un esquema para la prueba del control de aceleración. Este esquema será el mismo que se usará en la configuración final, a excepción de que no se necesitará el potenciómetro de consigna, ya que seráel software el encargado de ello exclusivamente en los momentos necesarios.

Se ven los filtros RC para cada una de las señales a clonar, la conexión a masa y el potenciómetro que se usa el el programa de prueba para asignar un valor de accionamiento del acelerador.

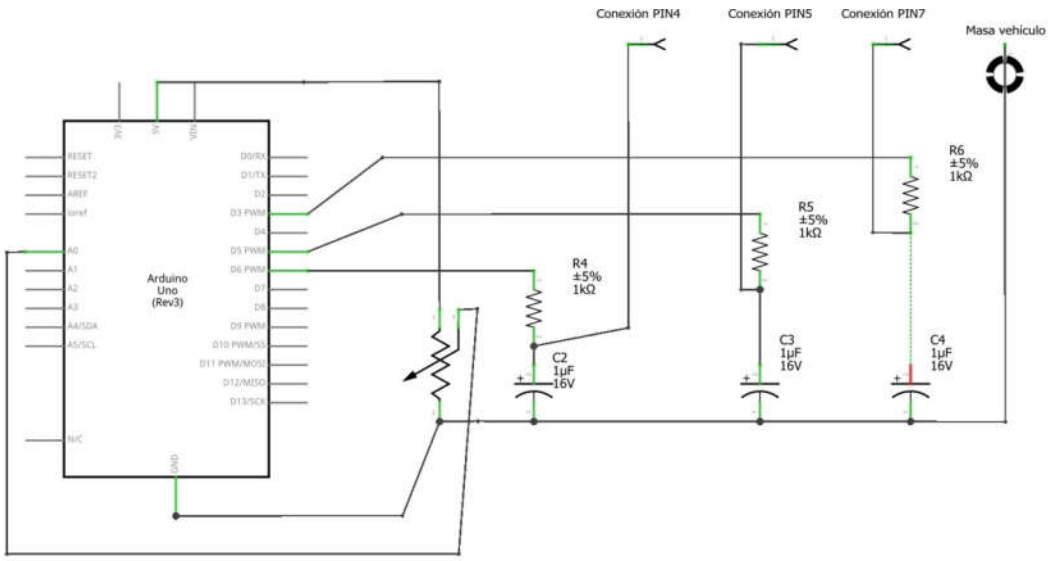


Ilustración 29: Esquema de montaje para la prueba de clonación de acelerador

Este boceto de montaje es el esquema anterior comentado e insertado en la documentación; de esta forma se hace la prueba de las conexiones necesarias, así como del código que se comentará más adelante.

Las conexiones a la protoboard son obvias y están claras, pero los Pads en gran escala corresponden a los pines 4, 5 y 7 del conector del acelerador, asi como el Pad de masa corresponde a la conexión a masa del vehículo.

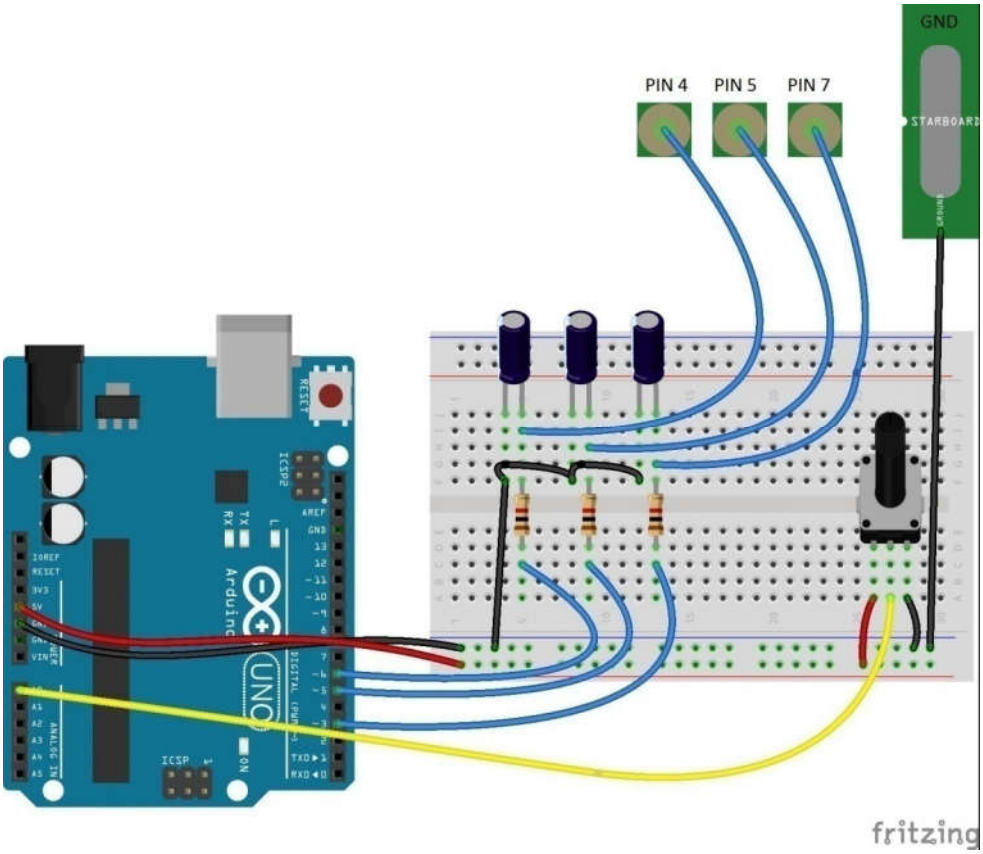


Ilustración 30: Montaje real para la prueba de clonación de acelerador

El resto de conexiones del conector del acelerador se quedan puenteadas con bananas para la prueba e inalteradas, así se evita que la unidad de control detecte algun errorcomo se puede ver en la imagen.

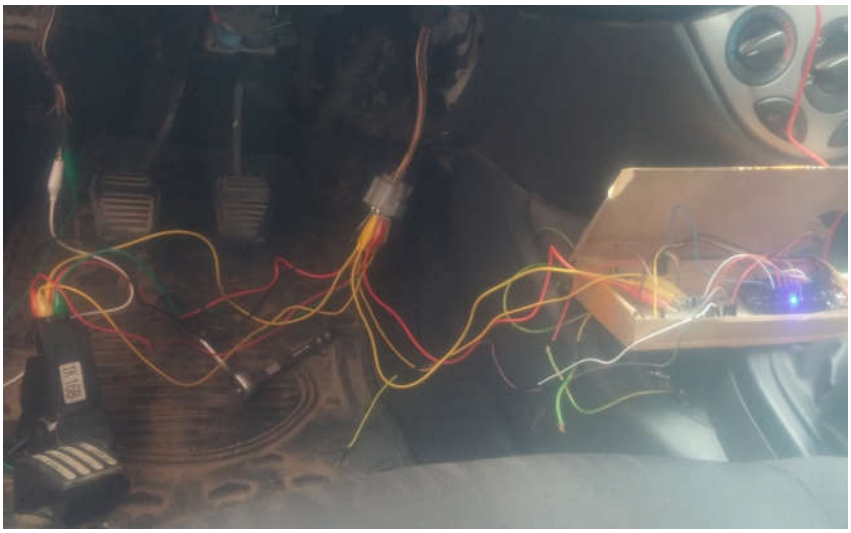


Ilustración 31: Prueba real de la clonación de acelerador

7.1.3 Código y resultados de la prueba de la clonación de señal de acelerador

El código empleado para la prueba también será el codigo emplado para el proyecto, al menos las partes más importantes que se explican a continuación. El código completo se encuentra en el documento *Anexo II: Código de Arduino.*

- Asignación de entradas y salidas.

```
constintanalogInPin= A0;  // Entrada analógica para el potenciómetro
de consigna
constint analogOutPin3 = 3; // Salidas PWM para la clonación del
acelerador
constint analogOutPin5 = 5;
constint analogOutPin6 = 6;

intsensorValue= 0;        //Asignación de valor inicial y tipo de
dato.
int outputValue3 = 0;
int outputValue5 = 0;
int outputValue6 = 0;
```

- Configuración de puerto serie: es a través de éste como se toman todos los datos en tiempo real a través de USB y el monitor de puerto serie. Esto se hace a través de la función void setup, que se ejecutara solo en el primer ciclo de progama.

```
voidsetup() {

  Serial.begin(9600);       // inicializa el puerto serie a velocidad
de 9600 bps
}
```

- Lectura de la señal analogica perteneciente al potenciómetro.

```
sensorValue=analogRead(analogInPin);
```

- Escalado y escritura de las salidas PWM
  Para el escalado de los límites de entrada se usan los valores de 0 a 1023, que corresponden a los valores de entrada anaógicos de 0 a 5V.
  Para el escalado de las señales de salida se ha usado la siguiente fórmula para los límites de salida, correspondiente a los valores mínimos y máximos de la tabla, con un rango de valores desde 0 a 255 (rango de valores con el que es posible configurar las salidas PWM).

$$Out\ Value = \frac{Vdc \quad 255}{5}$$

Pin	Acelerador 0% (Vdc)	Acelerador 100% (Vdc)	Acelerador 0% (OutValue)	Acelerador 100% (OutValue)
V4	1,29	3,9	66	199
V5	4,33	0,84	221	43
V7	0,7	3,33	36	171

Tabla 3: Datos para los límites en el escalado de las salidas

```
outputValue3 =map(sensorValue, 0, 1023, 66, 199); //65.79=1.29*255/5
pin 4
 outputValue5 =map(sensorValue, 0, 1023,221,43 ); // pin 5
 outputValue6 =map(sensorValue, 0, 1023, 36, 171); // pin 7
```

- Escritura de las salidas PWM

```
 analogWrite(analogOutPin3, outputValue3);
 analogWrite(analogOutPin5, outputValue5);
 analogWrite(analogOutPin6, outputValue6);
```

- Para obtener la lectura de los datos deseados se envían por el puerto serie a través de USB para la visualización y tratamiento de datos en el ordenador.

```
 Serial.print("sensor = ");
 Serial.print(sensorValue);
 Serial.print("\t");
 Serial.print(outputValue3);
 Serial.print("\t");
 Serial.print(outputValue5);
 Serial.print("\t");
 Serial.println(outputValue6);
```

- Tiempo de espera para comprobar cuál es el tiempo máximo de ciclo para no notar un retraso a la acción del acelerador, comparándolo con un comportamiento normal antes de la clonación. Como conclusión se observa un comportamiento con retraso con refrescos superiores a 200 milisegundos.

```
 delay(200);
}
```

En la siguiente figura se pueden observar las salidas escaladas correspondientemente y que la acción del acelerador es correcta.

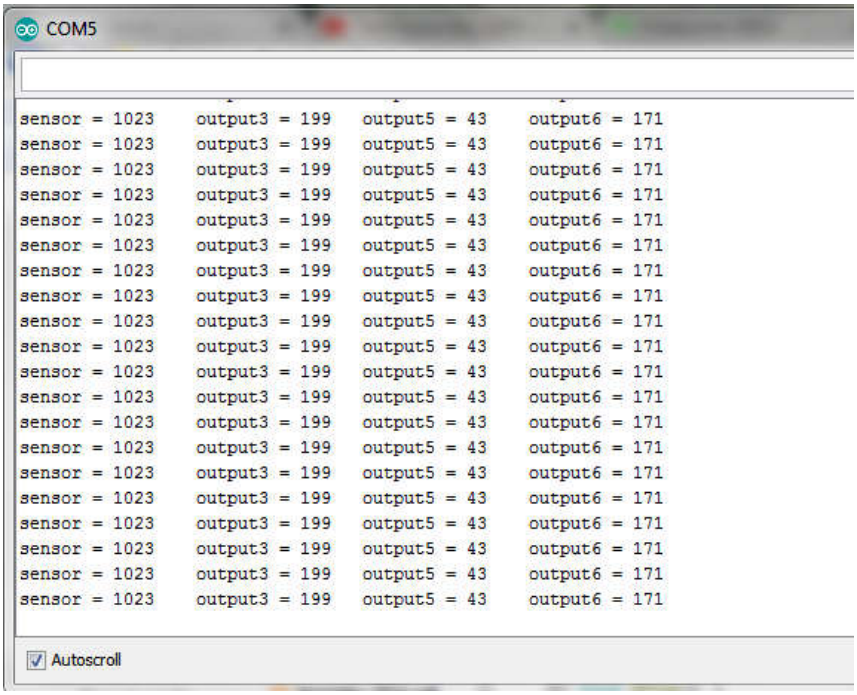


Ilustración 32: Datos reales de la prueba de clonación de acelerador

7.2 Adquisición de señales para la lectura de velocidad

Se han desarrollado cuatro formas diferentes para realizar la lectura de la velocidad real del vehículo, aunque solo una ha sido efectiva.

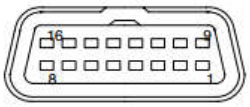
Todas ellas podrían ser válidas en otros proyectos, con otro tipo de vehículo o con otro tipo de componentes. En los siguientes apartadosse detallan las cuatros formas en las que se ha realizado la lectura de la velocidad.

7.2.1 Adquisición de señal para la lectura de velocidad a través del del CAN BUS del vehículo

La lectura de la velocidad del vehículo consiste en leer la velocidad haciendo un filtrado de los datos en el Can Bus del vehículo. Esta es la primera opción a tener en cuenta, tanto por su sencillez en la conexión, la gran cantidad de datos que se podrían obtener de la misma forma y lo poco invasiva que será para la aplicación deseada.

C200

F16016



Número del terminal	Descripción	Colores de cables
1	no se usa	
2	SCP+	GY/RD
3	no se usa	
4	Masa chasis	BK
5	Masa señal	BK/OG
6	CAN+	GY/RD
7	K–Línea ISO 9141	WH/BK
8	no se usa	
9	no se usa	
10	SCP–	BU/RD
11	no se usa	
12	no se usa	
13	Señal programación módulo	OG/BK
14	CAN–	BU/RD
15	no se usa	
16	Potencia batería	OG

Ilustración 33: Terminales del conector OBDII

En primer lugar se miden tensiones en los pines de comunicación.Únicamente se miden en los dos hilos del bus (CAN-H y CAN-L) obteniendo valores dentro de rango; dichos valores se encuentran entre 1.5 y 2.5 V para el hilo CAN-L y de 2.5 y 3.5 para el hilo de CAN-H

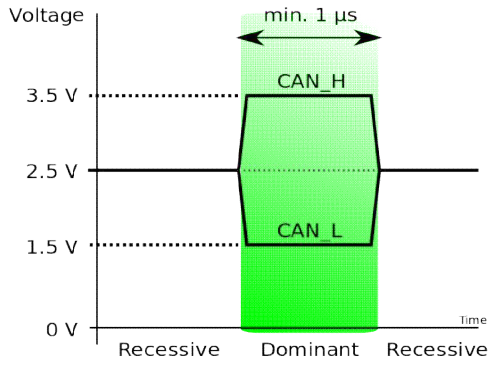


Ilustración 34: Tensiones en el Bus Can

Una vez comprobadas las tensiones se hace el montaje provisional del módulo con interfaz CAN y el microcontrolador Arduino para la adquisición de datos.

El módulo para la comunicación con el Bus se conecta directamente en los pines CAN-H y CAN-L, 6 y 14 respectivamente del conector OBD-II.

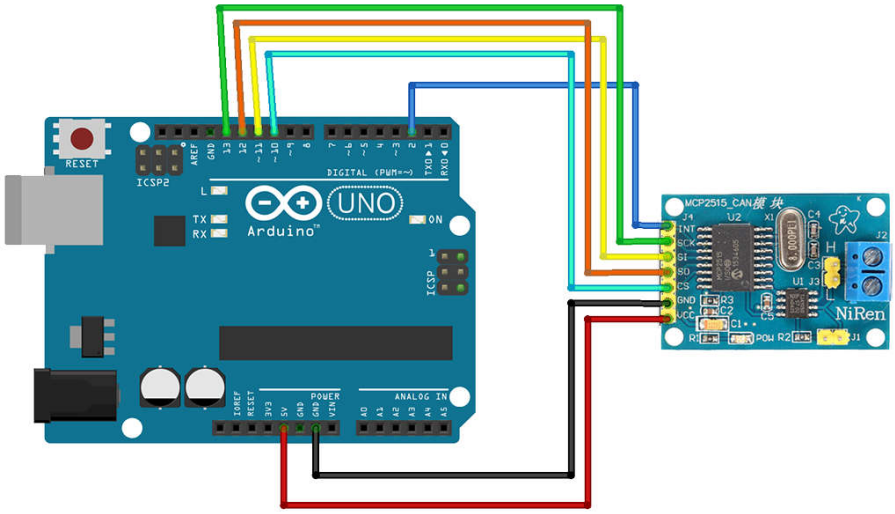


Ilustración 35: Conexión para la lectura del CAN Bus

De esta manera se desea interpretar la información que va por el CAN Bus monitorizando los datos a través de la comunicación serie por USB para más adelante seleccionar solo aquellos datos que se necesitan.

Se usa un pequeño programa para ver si se lee alguna trama del CAN Bus con ayuda de una librería de Arudino (MCP_CAN.h).

Este programa se encuentra en su versión completa en el documento *Anexo II: Código de Arduino.*

- Configuración de la comunicación serie para la lectura de datos con el ordenador.

```
Serial.begin(115200);
```

- Inicializa y configura el módulo de CAN.

```
if(CAN_OK==CAN.begin(CAN_500KBPS))
    {
        Serial.println("CAN BUS Shield init ok!");
```

- Imprime en pantalla los identificadores con sus datos asignados.

```
 if(CAN_MSGAVAIL==CAN.checkReceive())
{
    CAN.readMsgBuf(&len,buf);
    canId=CAN.getCanId();

      Serial.print(canId);Serial.print(",");
       for(inti= 0; i<len; i++){
          Serial.print(buf[i]);Serial.print(",");
       }
```

```
Serial.println();
```

Después de varias pruebas e intentos no se consigue leer ningún dato del Bus, aunque el cableado de Bus está instalado en el vehículo, lo más probable es que no disponga el mismo tipo de comunicación.

En los diagramas del cableado del vehículo no se encuentra información sobre el intercambio de información entre las distintas redes de comunicación, por lo que se hace imposible identificar la configuración de comunicación. Tampoco es posible encontrar información sobre el protocolo de comunicación del Ford Focus TDCI 114HP (85KW), obteniendo siempre protocolos PWM para modelos similares, pero ninguna información sobre el modelo del proyecto.

Model	Engine	Year (startingfrom)	OBD-2 Protocol
Ford Focus	1.6 16V, Gasoline (100HP)	2000	PWM
	1.8 TDCI, Diesel (114 HP)	2001	
	1.8 TDCI, Diesel (114 HP)	2001	
	1.4 16V, Gasoline (75HP)	2001	PWM
	Gasoline (100 HP)	2002	PWM J1850
		2002	PWM J1850
	Gasoline (171 HP)	2003	PWM J1850
	1.8 TDDI , Diesel (89 HP)	2003	
	1.8 tdCi, Diesel (100HP)	2003	PWM
	1.6 tdCi, Diesel (110HP)	2003	CAN 11bit (500kb)

Tabla 4: Protocolos de comunicación Ford Focus

Esto no es posible detectarlo por la escasa o nula información que existe de las características de comunicación del vehículo del proyecto, del Ford Focus TDCI 85kw DAW de 2002.

7.2.2 Adquisición de señal para la lectura de velocidad a través de sensor ABS.

Para la toma de velocidad en tiempo real se intenta tomar la señal de uno de los sensores de ABS, ya que de forma habitual se trata de sensores inductivos que proporcionan una señal senoidal de amplitud y frecuencia dependiente de la velocidad. Para aprovechar este sensor y su señal se decide utilizar un integrado que dará la señal convertida a una señal analógica lineal, este integrado es LM2917N.

En la documentación del integrado se dispone de muchas configuraciones, es este caso se elige una de ellas para conversión frecuencia-tensión.

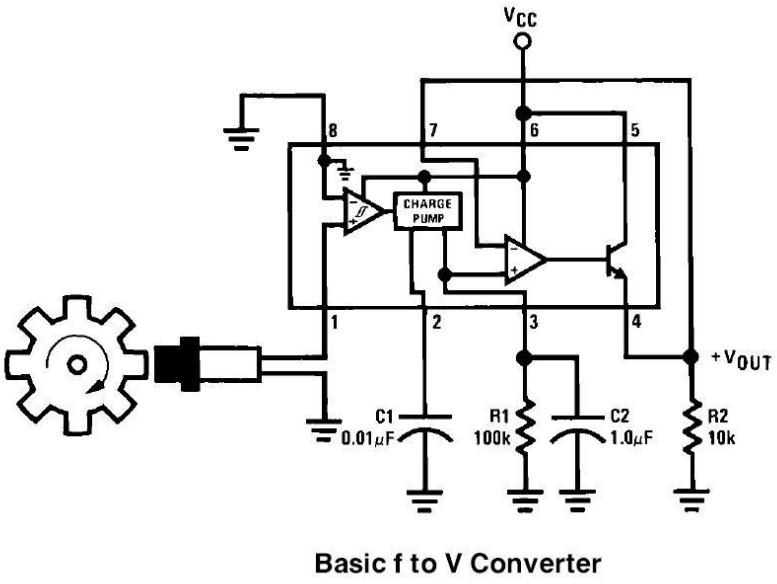


Ilustración 36: Montaje para conversión frecuencia tensión

Una vez se compruebe que ésta es una solución válida, se calcularán los componentes en base a las necesidades específicas del vehículo.

La salida con la señal convertida debiera tener una respuesta parecida a la figura siguiente.

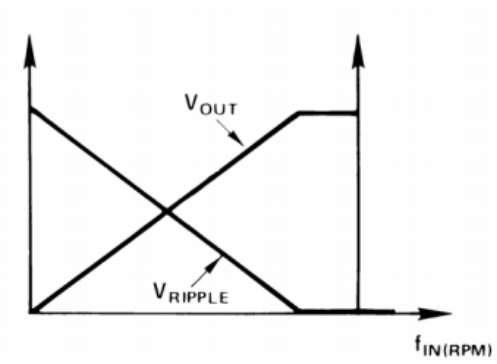


Ilustración 37: Respuesta f-V

Se realiza el montaje provisional en la protoboard para la prueba con el circuito que sugiere el fabicante, para a continuación hacer una busqueda de dicho sensor y las posibilidades de conexión real.

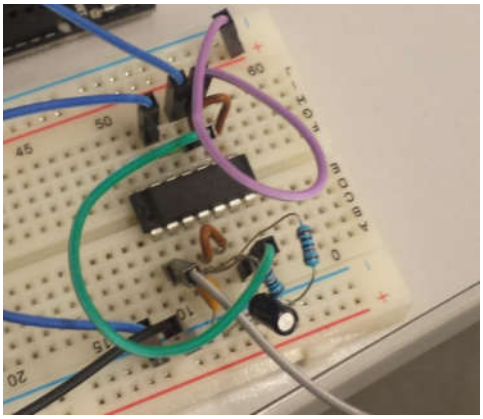


Ilustración 38: Montaje LM2917

Con el montaje de forma provisional en una protoboard se busca en la documentaciónd eléctrica del Ford Focus las conexiones de los sensores ABS y la situacion de los mismos para hacer las pruebas.

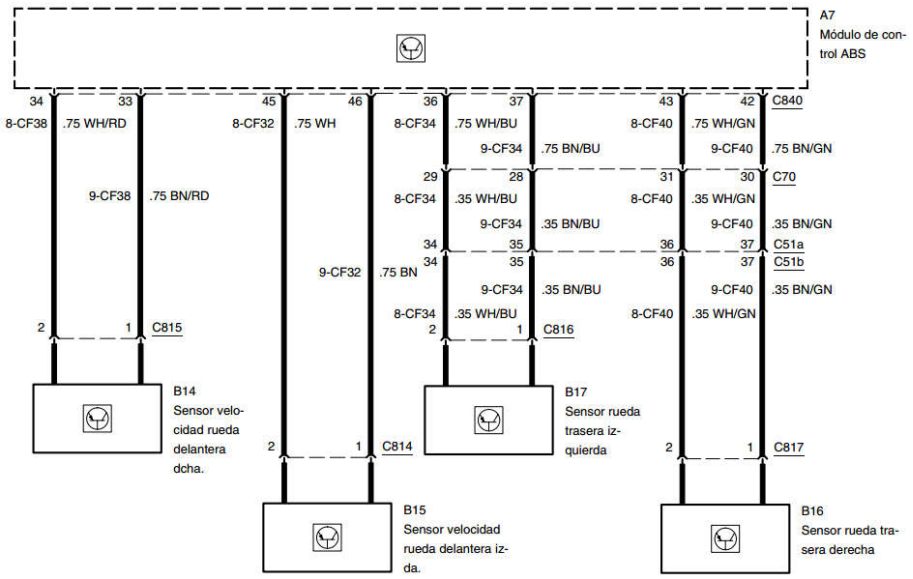


Ilustración 39: Conexión de los sensores ABS

Se ha eligido la rueda delantera izquierda tanto por su situacion como por su mejor accesibilidad, como se muestra en los esquemas eléctricos, esquemas que se encuentran en el *Anexo I: Esquemas eléctricos* con mayor detalle.

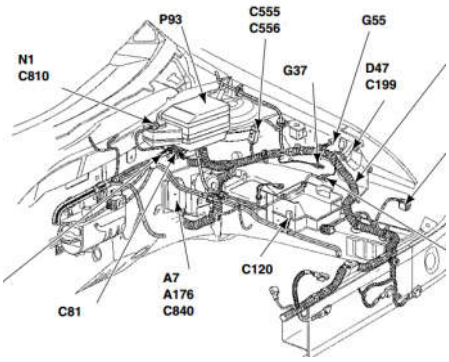


Ilustración 40: Situación de sensor delantero izquierdo de ABS

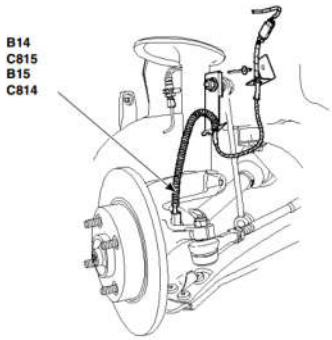


Ilustración 41: de sensor delantero izquierdo de ABS en la mangueta

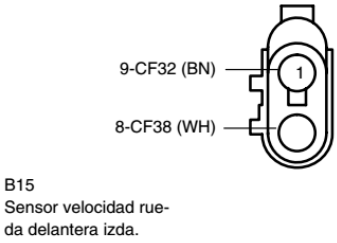


Ilustración 42: Conector de sensor delantero izquierdo de ABS

Ilustración 43: Conexión directa al sensor B15 de ABS

Tras el trabajo realizado y tras la comprobación de las conexiones, las pruebas no dan resultados. Después de varias comprobaciones se detecta que no se dispone en estevehículo de un sensor magnético, sino que se trata de un sensor ABS activo de una nueva generación de sensores de movimiento, con una comunicacón entre sensor y módulo de control de ABS. No se ha encontrado información suficiente sobre este tipo

de sensores, por lo que se ha desestimado esta opción para la lectura dela velocidad del vehículo.

Ilustración 44: Prueba de medida se señal en sensor ABS

7.2.3 Adquisición de señal para la lectura de velocidad a través de sensor de velocidad VSS

A través del sensor VSS del vehículo también se puede hacer una lectura de la velocidad del vehículo, de hecho, es el sensor que usa el mismo para hacer dicha lectura. En este caso, este sensor se encuentra acoplado a la transmisión de salida para la rueda derecha y se trata de un sensor magnético, haciendo esto posible usar la configuración que se había diseñado para la lectura con el sensor de ABS.

Para hacer esto posible se busca en la documentacion la situación y conexión de dicho sensor. En los planos eléctricos del vehículo se encuentra el sensor B11, dicho sensor es el de velocidad del vehículo, estos pastes de la documentacion de encuntran en el *Anexo I: Esquemas eléctricos* con mayor detalle.

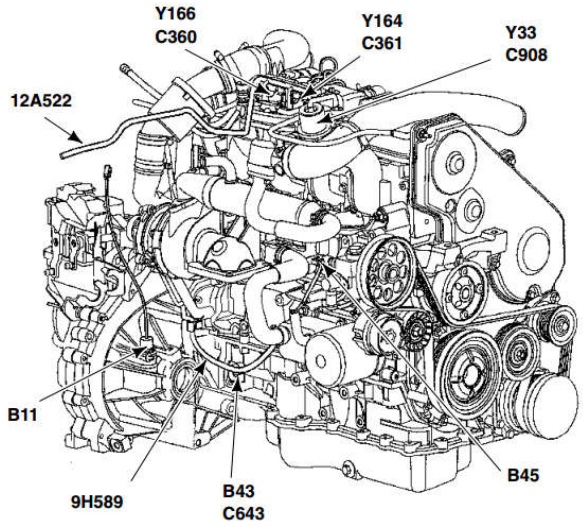


Ilustración 45: Ubicación del sensor VSS (B11)

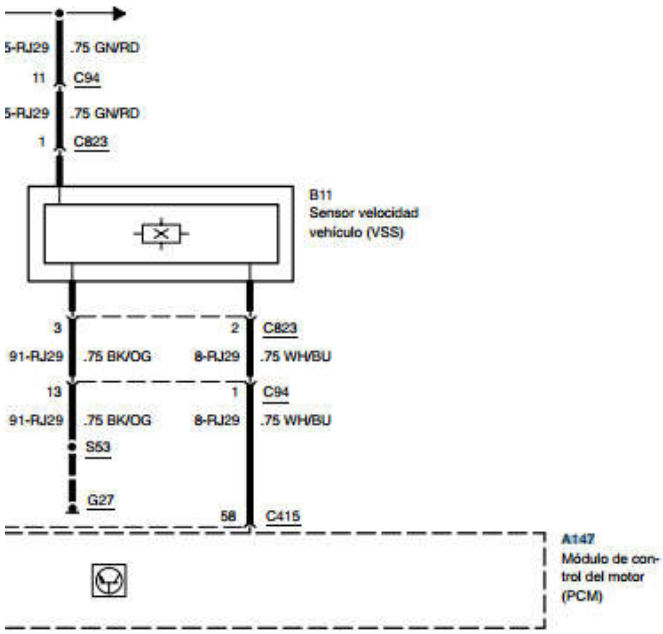


Ilustración 46: Conexiones del sensor VSS al módulo de control del vehículo

Aunque el sensor es válido para la prueba y la situación del mismo es relativamente accesible, se decide dejar esta opción para buscar otra menos invasiva, evitando así hacer conexiones en el cableado que va directo al módulo de control del vehículo.

7.2.4 Adquisición de señal para la lectura de velocidad a través de adquisición de las señales de alimentacion del indicador de velocidad.

Como alternativa a las anteriores, se decide sacar las señales del indicador de velocidad del cuadro de instrumentos.

En primer lugar se hace la busqueda en la documentación eléctrica del vehículo, aunque en este caso es muy escasa, obteniendo unicamente una representación simbólica del cuadro de instrumentos.

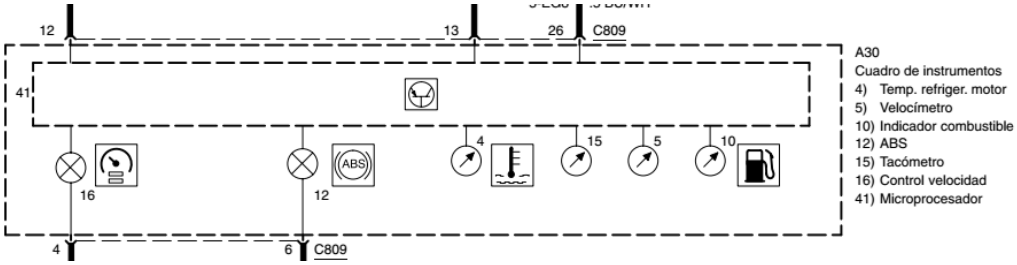


Ilustración 47: Esquema simbólico del cuadro de instrumentos

Esta falta de información hace necesario el desmonaje del cuadro y la busqueda de alguna posiblilidad de hacer la lectura de alguna señal válida para la adquisición de la velocidad.

Una vez desmontado el cuadro, se obtiene el despiece que se ve a continuación, obteniendo dos partes significativas: la placa de control y el bastidor con las cuatro bobinas de los cuatro indicadores.

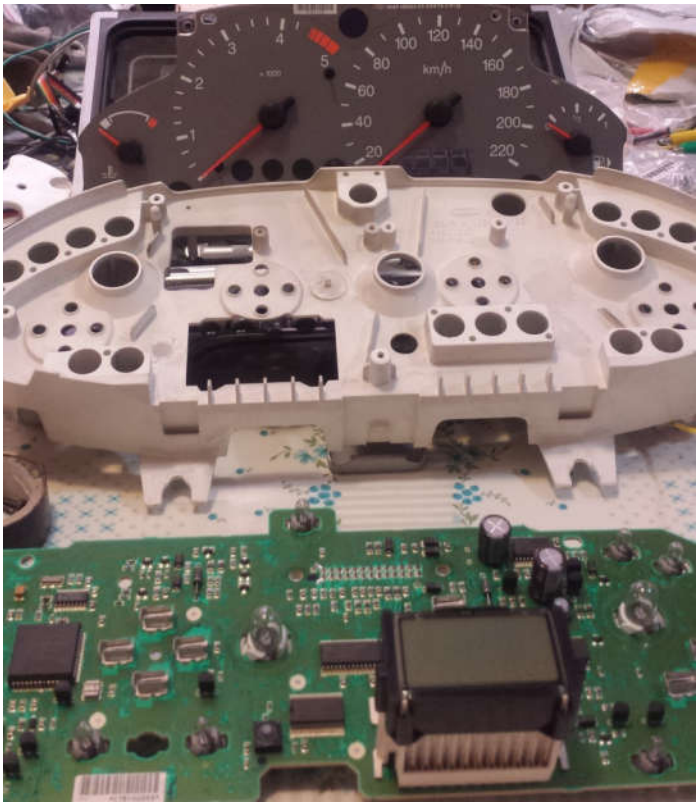


Ilustración 48: Despiece del cuadro de instrumentos

En la placa de control del cuadro de instrumentos se centra la inspección en aquello que da alimentación a las bobinas. Se diferencian cuatro circuitos integrados que dan alimentación.

Con la búsqueda de información de ese integrado no se ha encontrado nada, haciendo así necesaria la inspección de forma experimental.

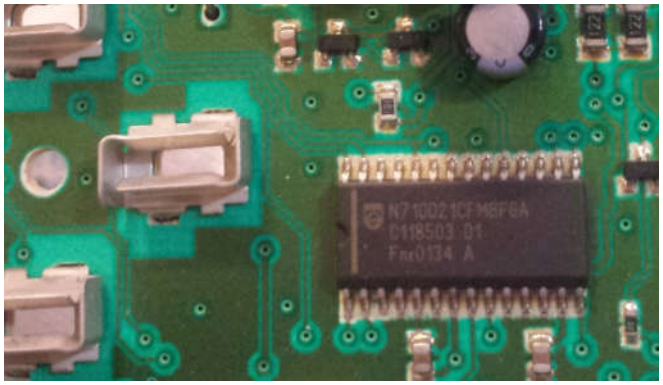


Ilustración 49: Integrado para los indicadores

Midiendo la resistencia de las bobinas (190 Ω) se obtiene su disposición y alimentando éstas con una fuente de corriente continua se analiza el comportamiento.

De esta forma se asegura que se trata de dos bobinas dispuestas perpendicularmente, con lo que con diferentes alimentaciones en sus cuatro terminales se cubren las posibilidades de los 360º del indicador.

Ilustración 50: Análisis del indicador

De esta forma se está seguro de que se puede obtener la velocidad del vehículo, que aunque se trata de una forma un tanto "bruta" de hacerlo, cumple mejor que ninguna la característica de poco invasiva en la electrónica del vehículo.

Para la realización, se extraen cuatro cables soldados directamente a las soldaduras de las conexiones de las bobinas y se miden las tensiones máximas de salida, para ver si hay necesidad de adaptar dichas señales. Todo esto se hace después de montar de nuevo el cuadro de instrumentos para que funcione con normalidad y obteniendo tensiones máximas de 10 voltios, por lo que será necesario adaptar dichas señales para el microcontrolador Arduino, que tiene como entrada máxima5 voltios.

Para la conexión al microcontrolador solo será necesario un simple divisor de tensión con resistencias iguales, se opta por dos resistencias de 5k ya que esta forma no se ve afectado en absoluto el funcionamiento del indicador que tiene 190 Ω.

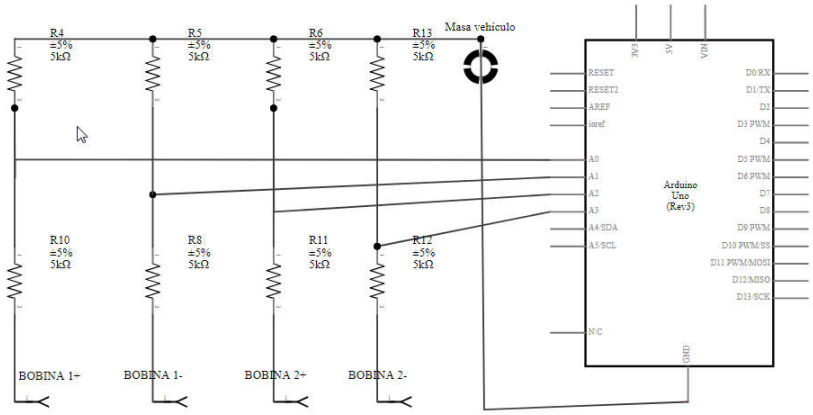


Ilustración 51: Circuito esquemático de la lectura del indicador de velocidad

Las pruebas de la adquisición de datos se llevan a cabo como en el resto del proyecto a través de la conexión provisional en al protoboard, quedando como se ve en el ejemplo.

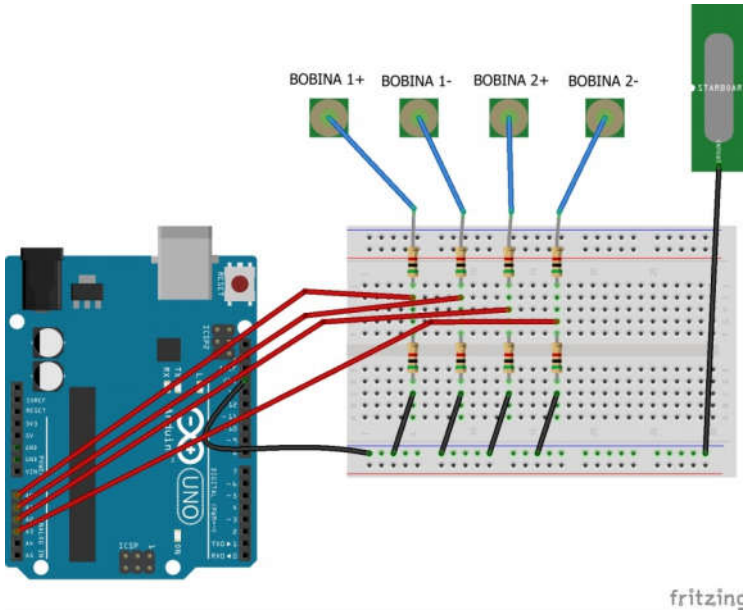


Ilustración 52: Conexión para las pruebas de la lectura del indicador de velocidad

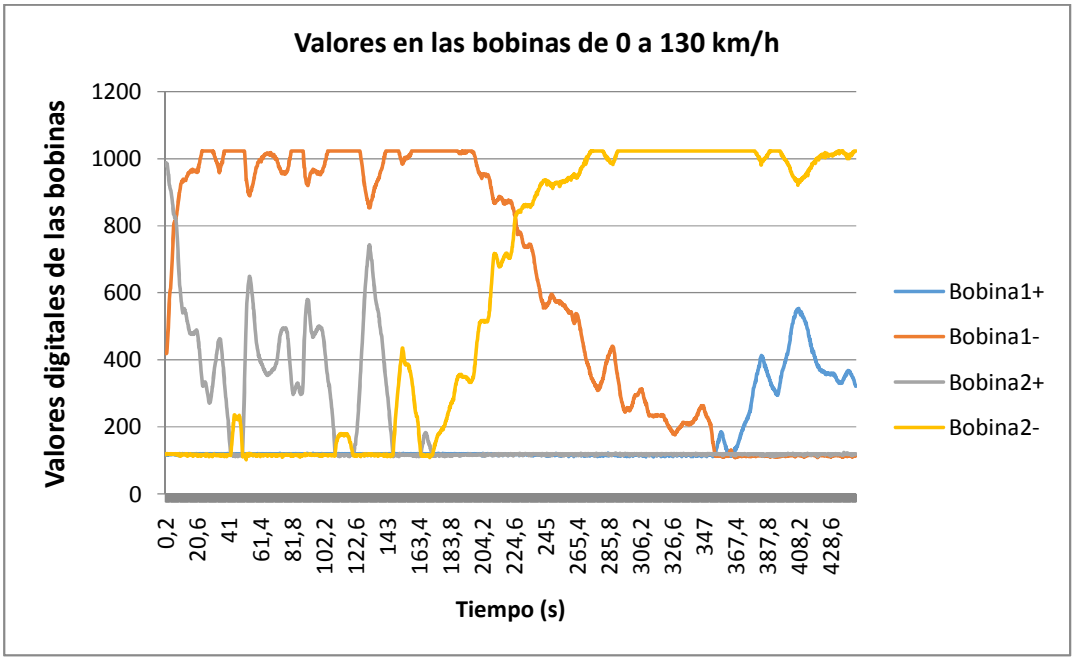
Con un sencillo programa en arduino, disponible en *Anexo II: Código de Arduino,* se obtienen los datos a través de USB y se representan para su análisis.

Estos datos tendrán un rango de 0 a 1023, que es el rango en el trabajan las entradas analógicas en Arduino para tensiones de 0 a 5 voltios.

119	**1023**	**274**	**116**
119	1023	274	116
119	1023	280	116
119	1023	286	115
118	1023	291	116
119	1023	301	115
118	1023	306	114
119	1023	319	116
117	1023	319	111

Tabla 5: Muestra de los datos recogidos con Arduino.

En el gráfico siguiente se representan los datos obtenidos en un trayecto acelerando hasta los 130 km/h. Se puede observar de forma clara como la respuesta de las bobinas es lineal y por tramos dependiendo del cuadrante en el que esté el indicador, y por lo tanto, de la velocidad.

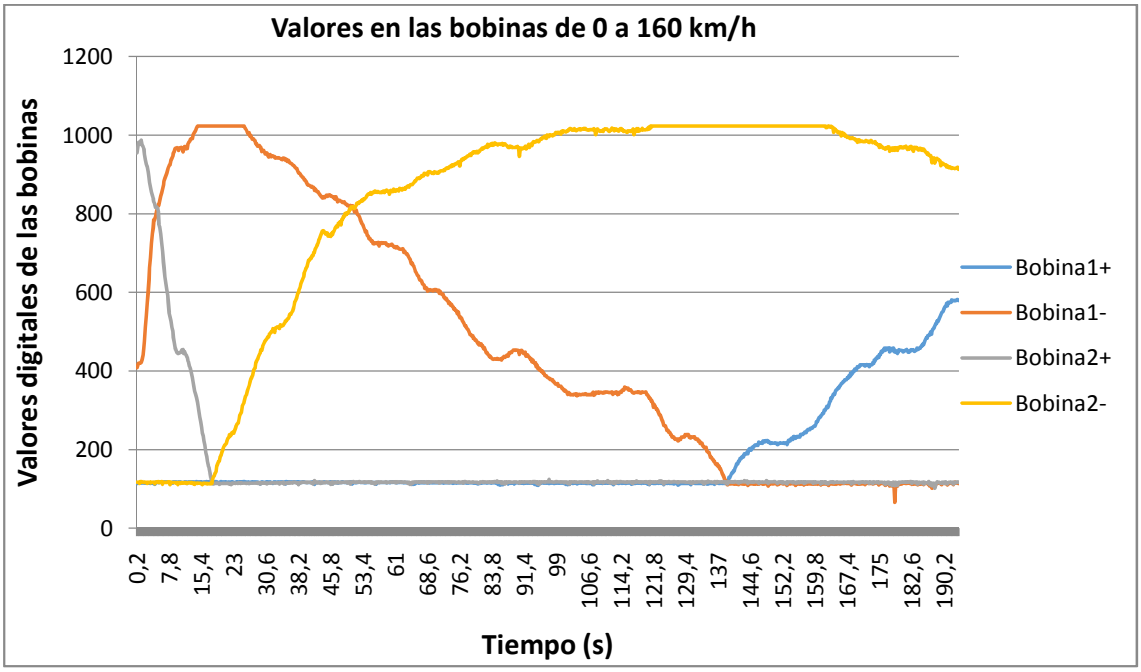


Gráfica 1: Prueba de adquisición de datos de velocímetro

Se toman las velocidades en los momentos que se obtienen las tensiones máximas y mínimas en las cuatro conexiones a las bobinas del indicador, obteniendo así sus pendientes y sus valores de corte.

Esta segunda interpretación se realiza con un aumento de velocidad lo más progresivo posible y sin desaceleraciones, obteniendo así unos resultados mucho más relevantes. En la siguiente gráfica se puede observar el orden en el que se va a priorizar los datos adquiridos en las bobinas:

- Bobina 2+ > Bobina2-
- Bobina 2- < Bobina1-
- Bobina 1- > Bobina1+
- Bobina 1+ < Bobina2-

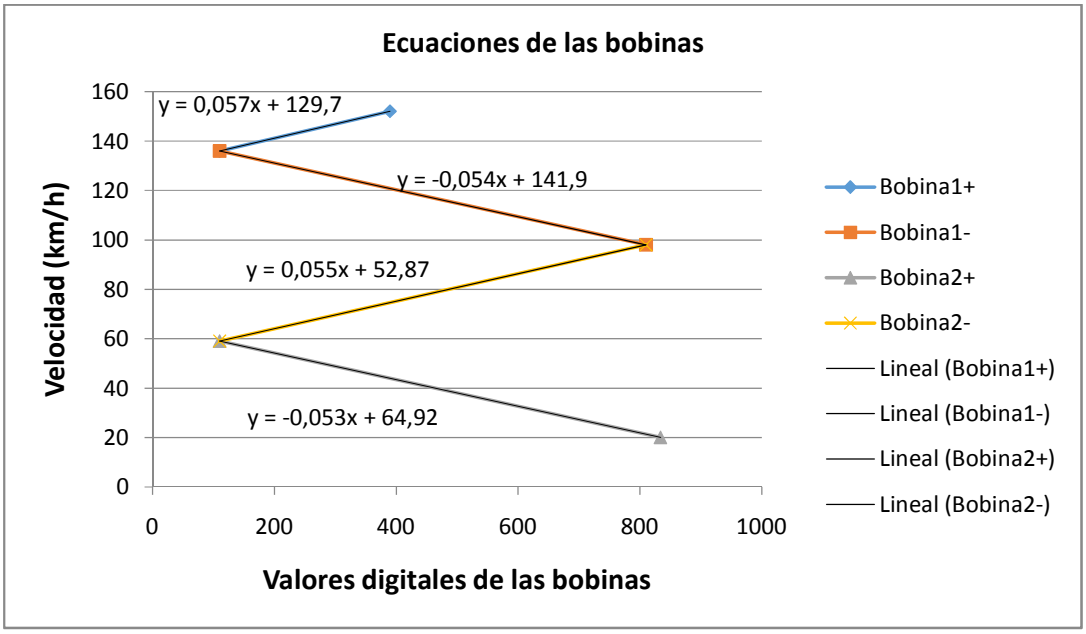


Gráfica 2: Adquisición de datos de 0 a 160km/h

Eligiendo los valores de corte como se ha comentado anteriormente, se calculan con la ayuda de Excel las ecuaciones lineales para así usarlas en el cálculo de la velocidad en el microcontrolador.

Bobina1+		**Bobina2-**		**Bobina2+**		**Bobina1-**	
Vdc	km/h	Vdc	km/h	Vdc	km/h	Vdc	km/h
390	152	110	59	834	20	110	136
110	136	810	98	110	59	810	98

Tabla 6: Valores de corte entre las ecuaciones



Gráfica 3: Tramos de las ecuaciones lineales

Como se puede apreciar todas las ecuaciones de las bobinas tienen pendientes prácticamente iguales, lo que indica que se han tomado los valores de referencia correctamente. Si esto no fuera así, sería muy sencillo ajustar dichos valores con la adquisición de datos de velocidad una vez se tenga instalando el GPS, ya que éste nos dará la velocidad con más precisión, sin aceleraciones ni desaceleraciones, manteniendo la velocidad constante.

7.2.4.1 Código y resultados de la prueba de lectura de velocidad del indicador.

Para representar la velocidad en tiempo real y visualizar si los cálculos y estimaciones son correctos se programa una función para representar en timpo real las velocidades en marcha.

A continuación se detallan las partes importantes del código, auque se dispone del programa completo en el *Anexo II: Código de Arduino.*

- Configuración de las entradas analógicas

```
constint analogInPin0 = A0;  // Configura entradas analógicas
constint analogInPin1 = A1;
constint analogInPin2 = A2;
constint analogInPin3 = A3;
```

- Configuración de puerto serie para la adquisición y visualización de datos

```
Serial.begin(9600);
```

- Lectura de las entradas analógicas

```
Bobina11 =analogRead(analogInPin0);
Bobina12 =analogRead(analogInPin1);
Bobina21 =analogRead(analogInPin2);
Bobina22 =analogRead(analogInPin3);
```

- Función para el cálculo de la velocidad, a través de las comparaciones comentadas en el apartado anterior y el uso de las ecuaciones para cada uno de los tramos.

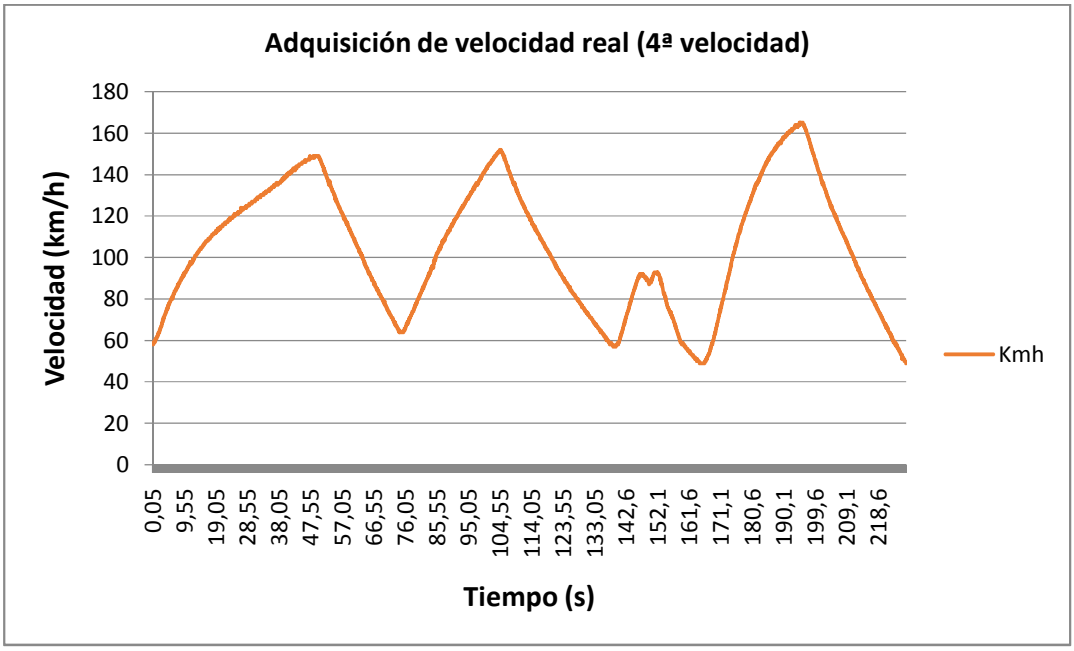
```
if ( Bobina21 > Bobina22) {
  vel= (-0.0539*Bobina21 + 65);
} elseif( Bobina22> Bobina21) {
      if ( Bobina22 > Bobina12) {
            if (Bobina11 > Bobina12) {
              vel= (0.0571*Bobina11 + 130);
            } else {
              vel= (-0.0543*Bobina12 + 142);
                              }
    } else {
      vel= ( 0.0557 * Bobina22 + 53);
    }

} else {
  vel= 200;
  }
```

- Envía por el puerto serie los datos

```
 Serial.print("sensor0 = ");
 Serial.print(Bobina11);
 Serial.print("\t");
 Serial.print(Bobina12);
 Serial.print("\t");
 Serial.print(Bobina21);
 Serial.print("\t");
 Serial.print(Bobina22);
Serial.print("\t");
 Serial.println(vel);
```

Haciendo varias pruebas, guardando y representando los datos, se observa que el funcionamiento es correcto, y este método será válido para el proyecto.

En la siguiente grafica se puede observar una prueba en cuarta velocidad.



Gráfica 4: Adquisición de velocidad real en 4ª velocidad

7.3 Adquisición de coordenadas GPS con el módulo NEO-6M

Para comprobar el funcionamiento del módulo GPS NEO-6M se monta un circuito provisional en una protoboard. Se trata de una conexión con alimentación a 5 voltios y una conexión de comunicación serie a 9800 baudios de velocidad.

Para la conexión de comunicación es necesaria una conversión de la señal RX, ya que el módulo trabaja con tensiones de 3.3 V, para el caso de la señal de TX no hay ningún problema ya que el microcontrolador Arduino interpreta correctamente a 3.3V una señal a nivel alto.

Se usan dos resistencias, una de 10k y otra de 4.7k, resultado del cálculo para el divisor de tensión.

Para tensiones a nivel alto de Arduino (5 V) llegaran señales a nivel alto al módulo GPS de 3.4 V, con lo que se asegura el correcto funcionamiento a largo plazo.

$$RX_{GPS} = 5 \quad \frac{10K}{10K + 4.7K} = 3.4\,V$$

En la figura siguiente se ve como queda el montaje para las pruebas del GPS.

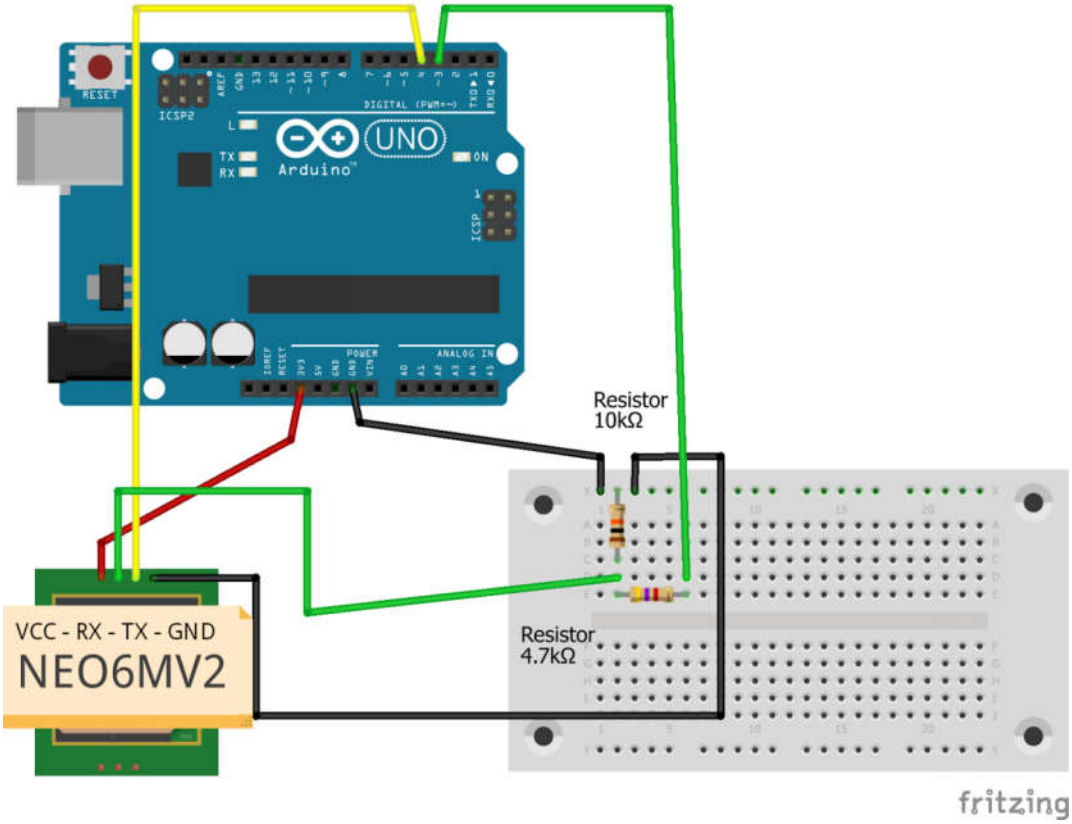


Ilustración 53: Montaje para pruebas GPS

Para ver su funcionamiento y decidir qué datos interesan se representan a través del puerto serie todas las tramas que envía el módulo de GPS, aún sin el uso de ninguna librería como más adelante se verá. El código completo se encuentra en *Anexo II: Código de Arduino.*

Como el Arduino UNO solo dispone de un puerto de comunicación serie y son necesarios dos en este caso, se habilitan los pines 3 y 4 con ayuda de una librería para un segundo puerto de comunicación serie. Esto no es recomendable utilizarlo ya que esta librería consume muchos recursos del microcontrolador, aunque en este caso para hacer las pruebas no generará ningún problema.

- Inserción de la librería para puertos serie adicionales:

```
#include<SoftwareSerial.h>//Libreria para generar otros pines adicionales como pines de comunicación
```

- Configuración del puerto adicional para el módulo GPS:

```
constint RX = 3;
constint TX = 4;
SoftwareSerialgps(RX, TX); // Asignación de pines para la comunicación GPS
```

- Configuración de las velocidades de transmisión en los diferentes puertos serie:

```
voidsetup()
{
   Serial.begin(115200); //Configura velocidad para el envío de datos
al ordenador
   gps.begin(9600); //Configura velocidad para la comunicación con el
módulo GPS
}
```

- Función para imprimir en pantalla todos los caracteres que se van recibiendo

```
voidloop()
{

   if (gps.available())
   {
      char data;
      data=gps.read();
      Serial.print(data);
   }
}
```

Una vez obtenidas los mensajes enviados por el módulo GPS, se analizan los datos para visualizar si son correctos y decidir cuáles de ellos son necesarios para la aplicación y cuáles pueden ser de interés también par mejoras y perfeccionamiento del sistema.

```
$GPGSV,3,3,09,32,17,222,35*43
$GPGLL,4227.14506,N,00227.26400,W,201741.00,A,A*78
$GPRMC,201742.00,A,4227.14657,N,00227.27073,W,16.836,291.31,170119,,,A*47
$GPVTG,291.31,T,,M,16.836,N,31.180,K,A*34
$GPGGA,201742.00,4227.14657,N,00227.27073,W,1,07,1.55,394.1,M,49.9,M,,*45
$GPGSA,A,3,31,29,26,32,21,25,14,,,,,,3.16,1.55,2.75*0A
$GPGSV,3,1,09,02,15,040,09,12,,,39,14,27,241,34,21,25,168,36*4E
$GPGSV,3,2,09,25,49,078,44,26,22,288,30,29,77,047,39,31,57,299,38*7A
$GPGSV,3,3,09,32,17,222,35*43
$GPGLL,4227.14657,N,00227.27073,W,201742.00,A,A*7D
$GPRMC,201743.00,A,4227.14815,N,00227.27570,W,12.219,291.53,170119,,,A*4F
$GPVTG,291.53,T,,M,12.219,N,22.629,K,A*35
```

Ilustración 54: Tramas enviadas por el módulo GPS

Existen 8 tramas diferentes que envía el módulo GPS, de las cuales solo serán necesarios los datos de mensaje encabezado por $GPGGA, pero tampoco todos los datos.

Para decidir qué datos son necesarios se enumeran y se describe a que corresponde cada uno de ellos:

$GPRMC,201743.00,A,4227.14815,N,00227.27570,W,12.219,291.53,170119,,,A*4F

- 201743.00→ Hora UTC
- A → Estado del receptor OK
- 4227.14815, N → Latitud; 42º 27.14851 min, Norte
- 00227.27570, W →Longitud; 2º 27.2757 min, Oeste

- 12.219→ Velocidad en nudos
- 291.53→ Dirección de movimiento en grados
- 170119→ Fecha; 17 de Enero de 2019
- 4F→Checksum

De todos estos datos se decide extraer:

- La longitud y latitud, para el cálculo de proximidad a un radar fijo.
- El estado OK del receptor para saber si los datos están disponibles.
- La velocidad para la calibración más precisa de la lectura del indicador de velocidad si fuera necesario.
- La dirección en la que se está moviendo el vehículo para comparar la dirección en la que detecta el radar fijo.
- Checksum, para saber si existen nuevos datos o se repiten, por falta de señal o mala señal GPS.
-

7.3.1 Código y prueba de adquisición de coordenadas GPS a través de la librería TinyGPS.h

De la misma forma en la que se han representado en pantalla en el apartado anterior se representan en pantalla solo los datos elegidos y convertidos a través de la librería TinyGPS.

En esta prueba se mostrarán los datos más significativos, como la latitud, longitud y el funcionamiento correcto del GPS. Se puede consultar el código completo en el documento *Anexo II: Código de Arduino.*

- Incluye la librería GPS:

```
#include<TinyGPS.h>
```

- Se guarda cada secuencia recibida para trabajar con la librería.

```
while (softSerial.available())
 {
    char c =softSerial.read();
    if (gps.encode(c)) // Nueva secuencia recibida
       newData=true;
 }
```

- Se extraen y envían por el puerto serie los datos con ayuda de la librería.

```
if (newData)
 {
    float flat,flon;
    unsignedlong age;
    gps.f_get_position(&flat,&flon,&age);
    Serial.print("LAT=");
```

```
      Serial.print(flat  ==TinyGPS::GPS_INVALID_F_ANGLE?0.0  :  flat,
6);
      Serial.print(" LON=");
      Serial.print(flon==TinyGPS::GPS_INVALID_F_ANGLE?   0.0   :flon,
6);
      Serial.print(" SAT=");
      Serial.print(gps.satellites()
==TinyGPS::GPS_INVALID_SATELLITES?0 :gps.satellites());
      Serial.print(" PREC=");
      Serial.print(gps.hdop()          ==TinyGPS::GPS_INVALID_HDOP?0
:gps.hdop());
   }
```

Como se puede ver de una forma relativamente sencilla se obtienen los datos necesarios, más adelante con la unión de las distintas partes se añadirán los datos que sean necesarios.

```
LAT=42.451953 LON=-2.451183 SAT=8 PREC=97 CHARS=6420 SENTENCES=30 CSUM ERR=0
LAT=42.451953 LON=-2.451182 SAT=8 PREC=97 CHARS=6847 SENTENCES=32 CSUM ERR=0
LAT=42.451953 LON=-2.451180 SAT=8 PREC=97 CHARS=7275 SENTENCES=34 CSUM ERR=0
LAT=42.451953 LON=-2.451179 SAT=8 PREC=97 CHARS=7703 SENTENCES=36 CSUM ERR=0
LAT=42.451957 LON=-2.451179 SAT=8 PREC=97 CHARS=8131 SENTENCES=38 CSUM ERR=0
LAT=42.451957 LON=-2.451178 SAT=8 PREC=97 CHARS=8559 SENTENCES=40 CSUM ERR=0
LAT=42.451961 LON=-2.451177 SAT=8 PREC=97 CHARS=8987 SENTENCES=42 CSUM ERR=0
LAT=42.451961 LON=-2.451177 SAT=8 PREC=97 CHARS=9415 SENTENCES=44 CSUM ERR=0
```

Ilustración 55: Prueba de datos GPS a través de la librería TinyGPS

7.4 Prueba de funcionamiento del módulo Bluetooth.

Para comprobar el funcionamiento del módulo Bluetooth se monta un circuito provisional en una protoboard. Se trata de una conexión con alimentación a 5 voltios y una conexión de comunicación serie a 9800 baudios de velocidad.

Para la conexión de comunicación es necesaria una conversión de la señal RX, ya que el módulo trabaja con tensiones de 3.3 V, para el caso de la señal de TX no hay ningún problema ya que el microcontrolador Arduino interpreta correctamente a 3.3V una señal a nivel alto.

Se usan dos resistencias, una de 10k y otra de 4.7k, resultado del cálculo para el divisor de tensión. Para tensiones a nivel alto de Arudino (5 V) llegaran señales a nivel alto al módulo GPS de 3.4 V, con lo que se asegura el correcto funcionamiento a largo plazo.

$$RX_{BLUE} = 5 \quad \frac{10K}{10K + 4.7K} = 3.4\,V$$

En la figura siguiente se ve como queda el montaje para las pruebas del Bluetooth.

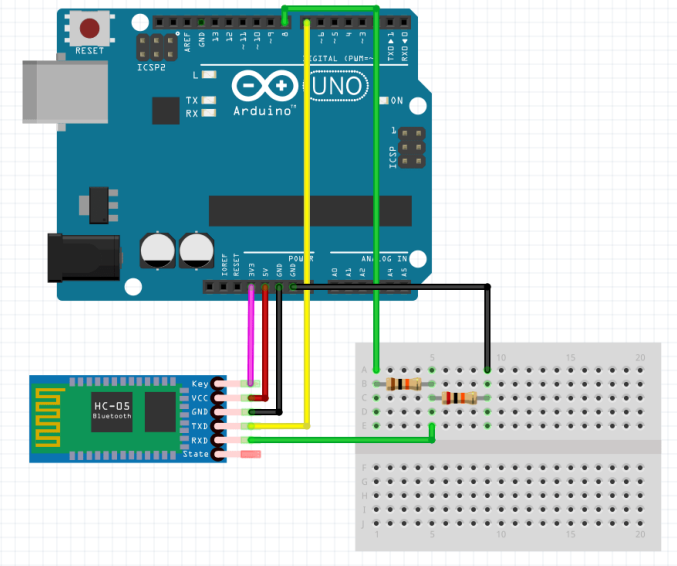


Ilustración 56: Montaje para la prueba del módulo Bluetooth

7.4.1 Código y prueba de funcionamiento del módulo Bluetooth

Para comprobar su correcto funcionamiento se utiliza un sencillo programa con el que se encenderá un led del Arduino conectado al pin 13 y se recibirán unos datos para representar en el Smartphone.

Con este mismo programa, el cual se encuentra de forma completa en el *Anexo II: Código de Arduino*, se prueba también de forma simultánea la App_v1 de este proyecto.

- Configuración de la velocidad de comunicación y la salida para el encendido del led.

```
voidsetup(){
  pinMode(13,OUTPUT);
  Serial.begin(9600);
}
```

- Cada vez que se reciba un carácter del módulo bluetooth se configura una acción. Es de esa forma como se van a dedicar determinados caracteres para diferentes acciones en el programa del microcontrolador.

```
voidloop(){
if(Serial.available()>0){              //  Si  el  puerto  serie  esta
habilitadp
        char_blue=Serial.read();        //  Lee  lo  que  llega  por  el
puerto Serie
}
```

```
if(char_blue=='1'){                    // on/off de pin 136
      digitalWrite(13,HIGH);
 }
if(char_blue=='2'){                    // on/off de pin 13
      digitalWrite(13,LOW);
 }
```

- De esa misma forma, se asigna el carácter “R” para la solicitud del envío de datos por parte de la App del Smartphone, así como el carácter “|” le indica a la App que ha finalizado el dato enviado.

```
  if(char_blue=='R'){          //  envía  todos  los  datos  cuando  la
aplicación lo solicita
  analogo=analogRead(A0);
  Serial.print(analogo);
  Serial.print("|");
  Serial.print("234");
  Serial.print("|");
  Serial.print("987");
  Serial.print("|");
  Serial.print(analogo);
  Serial.print("|");
  Serial.print("234");
  Serial.print("|");
  Serial.print("987");
  Serial.print("|");
  }
char_blue=0;
```

7.5 Adaptación del mando de Radio-CD

Para el control de velocidad, así como la activación y desactivación de funciones se ha aprovechado el mando del Radio-CD que existía en el equipamiento de serie del vehículo. Se trata de un mando con cinco botones, más de los necesarios, así que es la opción más interesante, sobre todo por su posición al alcance de las manos en el volante, como si de un mando de control de crucero original se tratase.

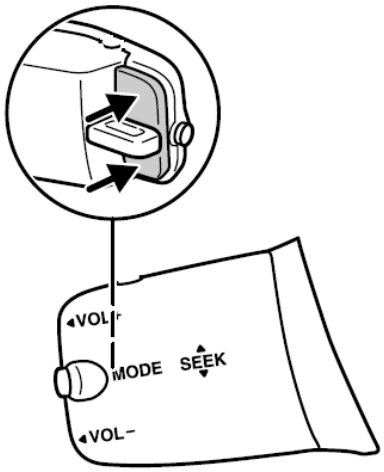


Ilustración 57: Mando radio-cd

Desmontando el mando se observa que solo existen dos cables conectados, estos van hasta el conector del antiguo radio-cd. Midiendo su resistencia se puede suponer su funcionamiento, cambiando su resistencia entre sus dos cables de salida con los distintos pulsadores.

Anotando dichas resistencias se diseña el esquema interior a través de ingeniería inversa.

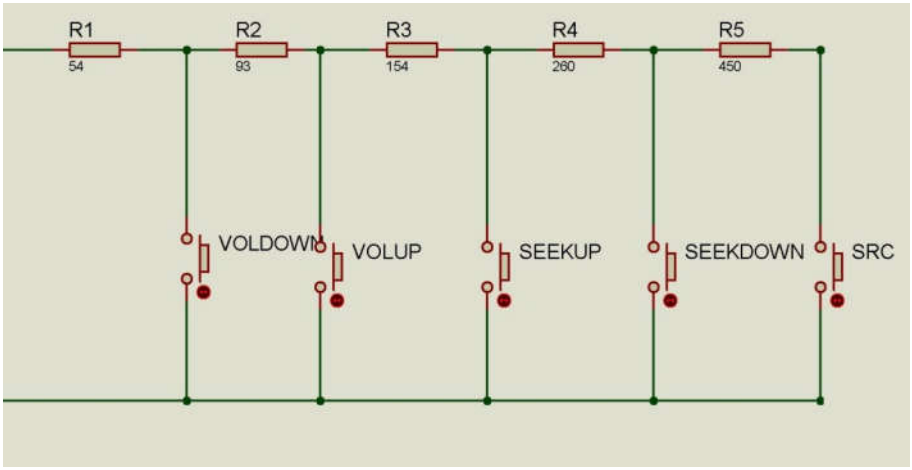


Ilustración 58: Esquema de mando radio-cd

Los valores medidos indican que su disposición y conexiones son esas, así como la prioridad de los botones desde VOL DOWN (Volumen menos) a SRC (mode). A esto se le añade una resistencia de 1KΩ para que junto al mando hacen un divisor de tensión que se alimenta con los 5 V del microcontrolador.

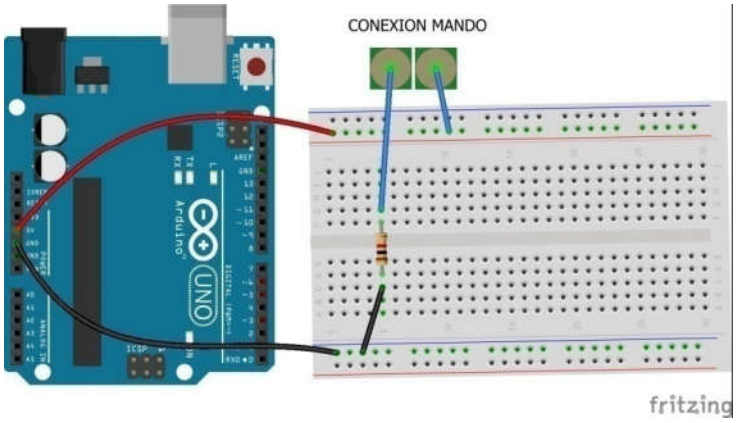


Ilustración 59: Prueba mando radio-cd

Con ayuda de cualquiera de los programas del proyecto se hace una lectura del los valores digitales de lectura del Arduino:

Boton	Resistencia (Ω)	Prioridad	Valor digital (Arduino)
Vol+	150	2	891
Vol-	54	1	972
Mode	1000	5	501
SK+	560	4	567
SK-	300	3	781
No Pulsado	5000	6	168
NC	-	-	0

Tabla 7: Datos mando radio-cd

7.5.1 Código para la identificación de los botones pulsados en Arduino

Con el microcontrolador Arduino se diferencian dos partes en la programación de la identificación de los botones pulsados.

A continuación, se describen las partes importantes del código, aunque la versión completa se encuentra en el *Anexo II: Código de Arduino.*

- En primer lugar se define el pin de entrada analógico procedente del divisor para el mando y los datos con los que se generan los anti-rebotes.

```
constint analogInPin6 = A6;

intvalor_mando= 0;
intcuentaVS=0;
intcuentaVB=0;
intcuentaSS=0;
intcuentaSB=0;
intcuentamode=0;
boolean P_VS =false;
```

```
boolean P_VB =false;
boolean P_SS =false;
boolean P_SB =false;
booleanP_mode=false;
interrormando= 1;
```

- Se configura la comunicación serie para la visualización de los botones pulsados.

```
voidsetup() {
  // initialize serial communications at 9600 bps:
       Serial.begin(115200);
}
```

- Lectura del valor analógico del mando.

```
valor_mando=analogRead(analogInPin6);
```

- En la siguiente función se compara el dato de entrada, y dependiendo de la prioridad antes comentada se aumenta el valor de la variable correspondiente por Ciclo de Scan, haciendo así la función de anti-rebote.

```
if (valor_mando< 100) { // Analiza botón pulsado y anti-rebote
errormando= 1;} //Error por mando desconectado
elseif (valor_mando< 300) { //Mando conectado
errormando= 0;
cuentaVS= 0;
cuentaVB= 0;
cuentaSS= 0;
cuentaSB= 0;
cuentamode= 0;
P_mode=false;
P_VS =false;
P_VB =false;
P_SS =false;
P_SB =false;}
elseif (valor_mando< 570) { // Pulsador mode
  cuentamode= cuentamode+1;}
elseif (valor_mando< 700) { //Pulsador Seek subir
  cuentaSS= cuentaSS+1;}
elseif (valor_mando< 820) { //Pulsador Seek bajar
  cuentaSB= cuentaSB+1;}
elseif (valor_mando< 930) { //Pulsador Velocidad subir
  cuentaVS= cuentaVS+1;}
elseif (valor_mando< 1020) { //Pulsador Velocidad bajar
  cuentaVB= cuentaVB+1;}
```

- En la siguiente función se compara el dato de cada una de las variables de cuenta y activa el valor del botón correspondiente.

```
if (cuentamode> 100)//Valida pulsacion
P_mode=true;
if (cuentaVS> 100)//Valida pulsacion
P_VS =true;
if (cuentaVB> 100)//Valida pulsacion
P_VB =true;
```

```
if (cuentaSS> 100)//Valida pulsacion
P_SS =true;
if (cuentaSB> 100)//Valida pulsacion
P_SB =true;
```

- Solo para la prueba se imprimen los datos y se visualiza si el botón pulsado corresponde con el que se pulsa en realidad.

```
Serial.print(P_mode);
Serial.print(P_VS);
Serial.print(P_VB);
Serial.print(P_SS);
Serial.print(P_SB);
Serial.println(valor_mando);

  delay(10);
  }
}
```

Los resultados son buenos y este mismo programa será la función correspondiente para la selección de los botones, sin programación adicional.

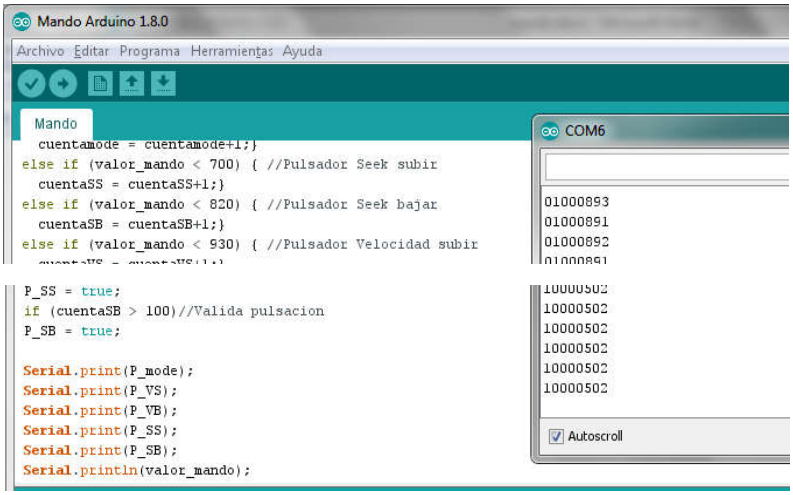


Ilustración 60: Prueba visual de la selección de botón pulsado

7.6 Diseño del conjunto de funciones y conexiones completas

En este punto del proyecto se decide ulilizar para el proyecto de forma definitiva la placa de desarrollo Aruino MEGA. Esta placa ofrece todas las características necesarias para el conjunto de funciones. La placa de desarrollo Mega cumple con las necesidades para las conexiones requeridas:

- 8 entradas analógicas: velocímetro, acelerador y mando Radio-CD.

- 3 salidas PWM para el control del acelerador.
- 3 puertos de comunicación serie: módulo GPS, módulo Bluetooth y modo monitor para la visualización de datos a través de PC.
- 2 entradas digitales para la conexión de freno y embrague.

Se puede consultar el esquema en el documento *Planos: Esquema de conexión Arduino MEGA.*

7.7 Diseño de la aplicación para el dispositivo móvil

Para el diseño de la aplicación móvil se ha usado APP Inventor, un entorno de programación libre y gratuito. Se trata de un entorno de desarrollo de aplicaciones para dispositivos Android, el cual funciona desde una plataforma web y solamente es necesario un navegador web.

Al construir las aplicaciones para Android se ha trabajado con dos herramientas: App Inventor Designer y App Inventor Blocks Editor.

En Designer se construye el Interfaz de Usuario, eligiendo y situando los elementos con los que interactuará el usuario y los componentes que utilizará la aplicación.

En el Blocks Editor se define el comportamiento de los componentes de tu aplicación.

7.7.1 Diseño del interfaz de usuario

Desde la herramienta App Inventor Designer, se diseñan los elementos necesarios para la visualización de los datos en tiempo real del sistema de control. Además de los elementos visuales se añaden también desde esta herramienta las funciones que sean necesarias.

Ilustración 61: Entorno de App Inventor

La siguiente imagen muestra el resultado de la apariencia que tendrá en un dispositivo móvil, diferenciando tres funciones diferentes:

- Conexión, desconexión y mensaje de texto para mostrar el estado de la conexión.
- Representación de seis datos numéricos para la visualización de datos en tiempo real.
- Botones para la activación de señales de salida y muestra de su estado de activación a través de color.

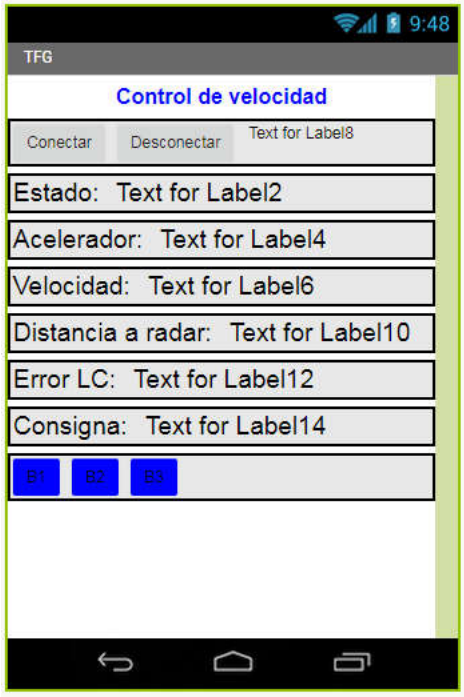


Ilustración 62: App. Interfaz para la visualización de datos

Para la comunicación con el microcontrolador son necesarias dos funciones añadidas:

- Una señal de reloj para configurar la comunicación en intervalos de tiempo, configura en intervalos de 250 milisegundos. Este valor de configuración se debe a que el tiempo de ejecución del programa en el Arduino no supera los 100 milisegundos, de esta forma se asegura la recepción de los datos cada vez que son solicitados.
- Función de conexión Bluetooth.

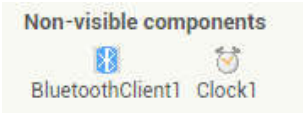


Ilustración 63: App. Funciones añadidas en la aplicación móvil

7.7.2 Programación de las funciones

Desde la herramienta App Inventor Blocks Editor se programan las distintas funciones para el interfaz gráfico. Esta programación se realiza a través de bloques de funciones, cuyas partes se explican a continuación:

- En primer lugar se inicializan los valores de todos los datos a representar.

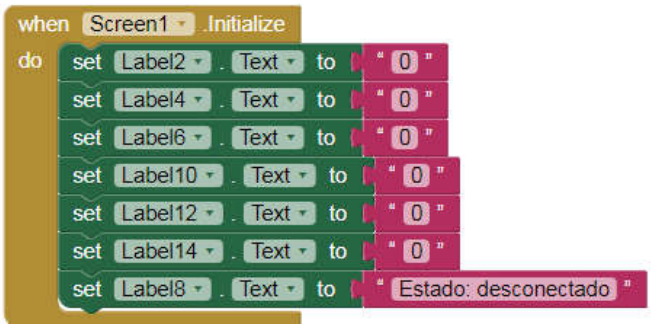


Ilustración 64: App. Inicializa etiquetas

- En el momento que se pulsa el botón de conexión, se activa la selección de una lista de dispositivos Bluetooth encontrados.

Ilustración 65: App. Listado de dispositivos

- Una vez elegido el dispositivo de conexión, se conecta el Bluetooth a ese dispositivo y se muestra un mensaje de la conexión o del error de conexión.

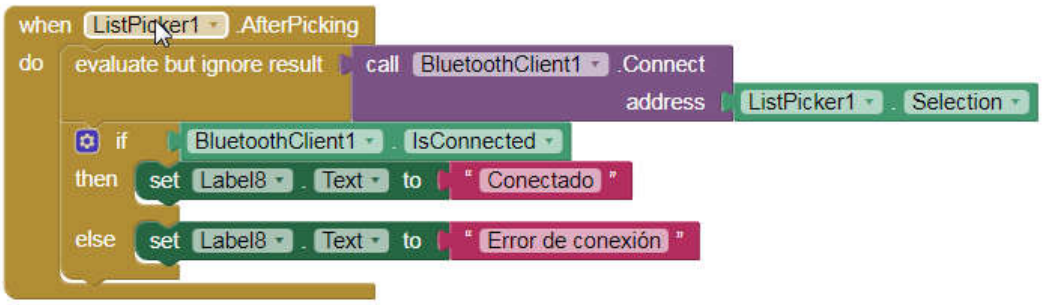


Ilustración 66: App. Conexión Bluetooth del dispositivo

- En el caso de pulsar alguno de los botones, quedan configurados para el envío de un carácter, el cual estará relacionado en el código de Arduino y a su vez cambiaran de color.

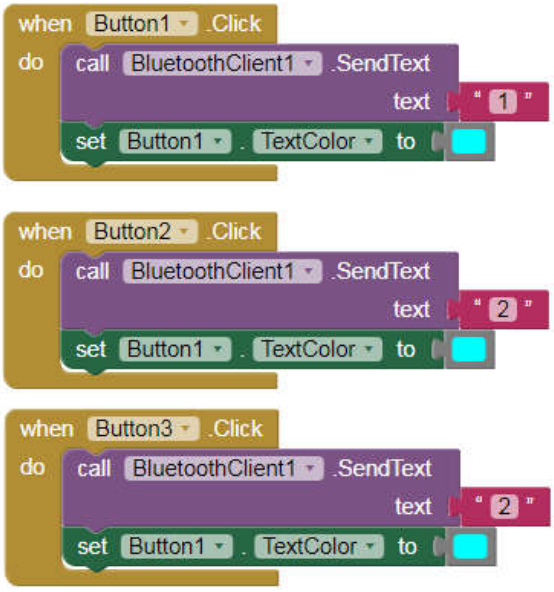


Ilustración 67: App. Pulsación de botones

- Para la visualización de datos en tiempo real se decide usar la aplicación móvil para elegirel momento en que se reciben los datos a través de la función añadida Clock 1, para el refresco de datos cada 250 milisegundos. De esta forma, en esta parte del código, primero se envía el carácter “R”, el cual identificará en el código de Arduino qué es lo que puede enviar. Todos los datos recibidos se introducen en una lista, diferenciando el cambio de dato a

través del carácter "|", el cual, Arduino enviará cada vez que finalice el envió de un dato. Por último, se actualizan los valores representados en las etiquetas de la interfaz gráfica.

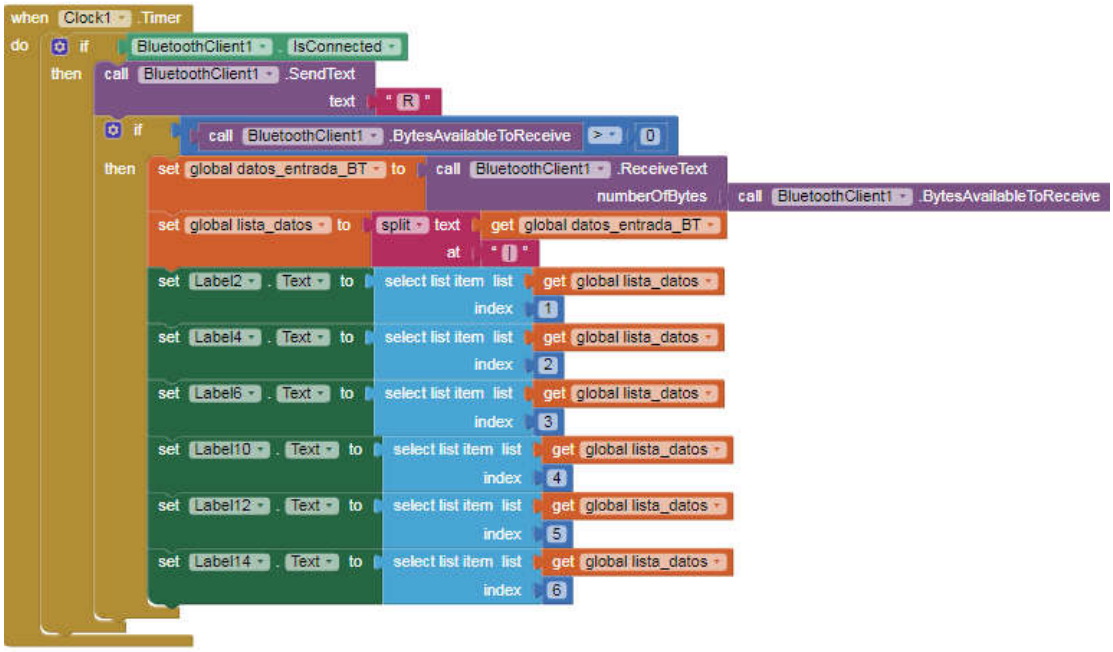


Ilustración 68: App. Representación de datos.

- En caso de necesidad de desconexión Bluetooth, será posible a través del botón "Desconexión".

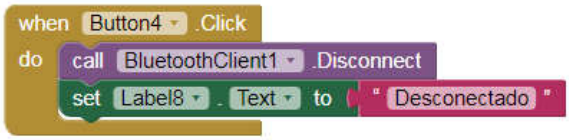


Ilustración 69: App. Desconexión Bluetooth

7.8 Código del programa completo en Arduino Mega

7.8.1 Programa principal

En este apartado se desglosa el programa final a través de la estructura resultante de la descripción de las partes importantes del código y de los diagramas. Después de

varias versiones y de ir incorporando las diferentes secciones que se han programado para las pruebas, se obtiene un programa estructurado de la siguiente manera:

- Función “mando()” → Anti-rebote y selección de botón pulsado.
- Función “gps()” → Extracción de los datos GPS necesarios.
- Función “bluetooth()” → Encargada de la comunicación con la App.
- Función “vel()” → Lecturas analógicas y cálculo de la velocidad del vehículo.
- Función “acel()” → Lectura y clonación de la señal de acelerador.
- Función “loop()” → Programa principal que integra la máquina de estados para la gestión de los diferentes controles y encargado de llamar a las diferentes funciones dependiendo de las necesidades.

En este mismo programa existen dos funciones para la seguridad, las cuales no tendrá control la máquina de estados, estas funciones se encargan de llevar a un modo seguro el microcontrolador.

- Función “tiemposcan()”→ Se encarga de una forma periódica de comprobar si no se ha sobrepadado el tiempo de ejecución del programa principal.
- Función “whachdog()”→ Se encarga de una forma periódica de comprobar si el microcontrolador no se ha bloqueado.

7.8.2 Programa de la máquina de estados

Desde el programa principal se integra la máquina de estados para la gestión de los diferentes modos de control, que constan:

- Modo 1: Limitador de velocidad por proximidad a radar fijo, detectado por coordenadas GPS.
- Modo 2: Limitador de velocidad por proximidad a radar fijo, detectado por coordenadas GPS y limitador de velocidad con asignación de consigna manual.
- Modo 3: Limitador de velocidad por proximidad a radar fijo, detectado por coordenadas GPS y control de velocidad con asignación de consigna manual o como se conoce de forma común, control de crucero.
- Modo 4: Control de velocidad con asignación de consigna manual ocontrol de crucero.
- Modo 5: Limitador de velocidad con asignación de consigna manual.

Esta máquina de estados se ocupa de las funciones importantes y sensibles para la seguridad, como son la gestión y control de los modos de funcionamiento.

En el diagrama siguiente se puede observar cómo se asigna el último modo de funcionamiento memorizado, con una pulsación larga del botón “Mode”del mando de control situado en la proximidad del volante.

Para el cambio de modo tienen asignada la función los botones de mando “Seek up” y “Seek down”, con estos botones se cambia al modo anterior o al modo siguiente. Cualquiera de estos modos será desactivado cuando se pulse el freno, o el embrague o una pulsación corta del botón “Mode” del mando de control situado en la proximidad del volante. El código de la máquina de estados se encuentra en el *Anexo II: Código de Arduino.*

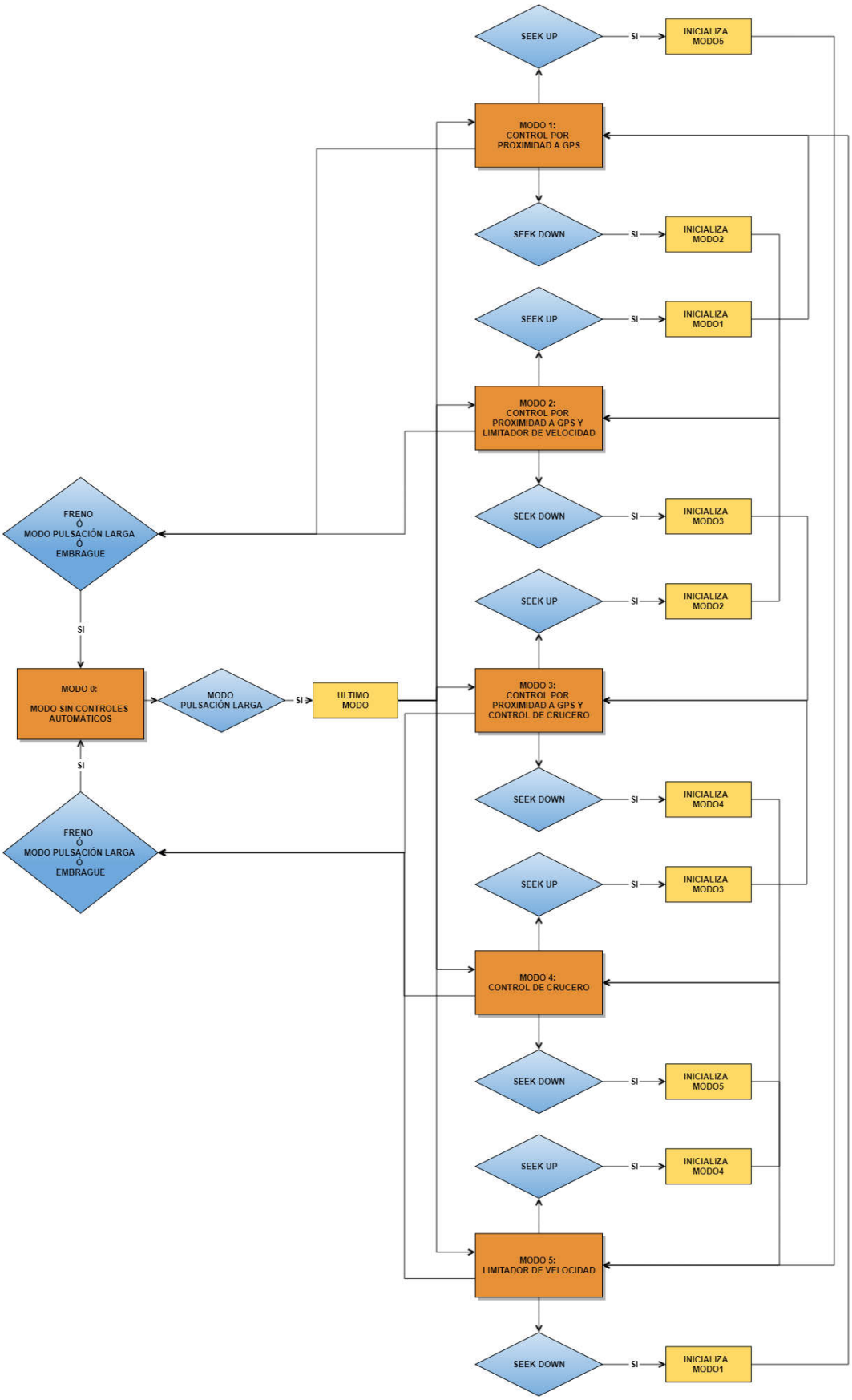

SEEK UP
SI
INICIALIZA MODO5
MODO 1: CONTROL POR PROXIMIDAD A GPS
SEEK DOWN
SI
INICIALIZA MODO2
SEEK UP
SI
INICIALIZA MODO1
MODO 2: CONTROL POR PROXIMIDAD A GPS Y LIMITADOR DE VELOCIDAD
SEEK DOWN
SI
INICIALIZA MODO3
SEEK UP
SI
INICIALIZA MODO2
MODO 3: CONTROL POR PROXIMIDAD A GPS Y CONTROL DE CRUCERO
SEEK DOWN
SI
INICIALIZA MODO4
SEEK UP
SI
INICIALIZA MODO3
MODO 4: CONTROL DE CRUCERO
SEEK DOWN
SI
INICIALIZA MODO5
SEEK UP
SI
INICIALIZA MODO4
MODO 5: LIMITADOR DE VELOCIDAD
SEEK DOWN
SI
INICIALIZA MODO1
FRENO Ó MODO PULSACIÓN LARGA Ó EMBRAGUE
SI
MODO 0: MODO SIN CONTROLES AUTOMÁTICOS
MODO PULSACIÓN LARGA
SI
ULTIMO MODO
SI
FRENO Ó MODO PULSACIÓN LARGA Ó EMBRAGUE

Ilustración 70: Diagrama de la máquina de estados.

En primer lugar, el programa comprueba que no se haya accionado ni el freno, ni el embrague ni la pulsación corta de "Mode", en cuyo caso los sistemas automáticos quedarán deshabilitados.

```
 if (freno ==true){
  acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
  estado= 0;}

  if (embrague ==true){
  acelerador_out = acelerador_in;//Señal de clonación igual a señal
real de acelerador.
  estado= 0;}

  if (P_mode_C ==true){  //Desconexión de sistemas automaticos
  acelerador_out = acelerador_in;
  estado= 0;}
```

En la estructura del programa en la máquina de estados se encuentra la selección de los modos, pero también a su vez la configuración inicial de dicho modo. A continuación en el siguiente fragmento del código se comentan la parte del código en el que se cambia al modo 2 o al modo 5 y se asignan las condiciones iniciales. Se puede consultar el código completo en *Anexo II: Código de Arduino.*

```
   if (P_SS ==true){  //Cambio de modo
    modo=2;
    set_lim_vel = vel; //Condiciones iniciales para el modo 2.
    estado= 2;}

    if (P_SB ==true){  //Cambio de modo
    modo=5;
setppoint= vel; //Condiciones iniciales para el modo 5.
    estado= 5;}
```

7.8.2.1 Modo 1: Limitador de velocidad por GPS

La función de este modo se trata de adecuar la velocidad del vehículo en el momento que éste se aproxime a un radar fijo o cualquier punto que se quiera añadir en el programa, como pueden ser cruces peligrosos o puntos de alta siniestralidad.

Como la desaceleración no es la misma en todo el rango de velocidades a controlar, en este modo se compara la distancia al punto más cercano, con el resultado de multiplicar la velocidad por un factor constante (frenaradar = vel * K_frenaradar), de esta forma el vehículo no regulará su velocidad demasiado tarde en velocidades altas o demasiado pronto en la velocidades bajas.

En la siguiente imagen se puede ver la estructura de dicho modo:

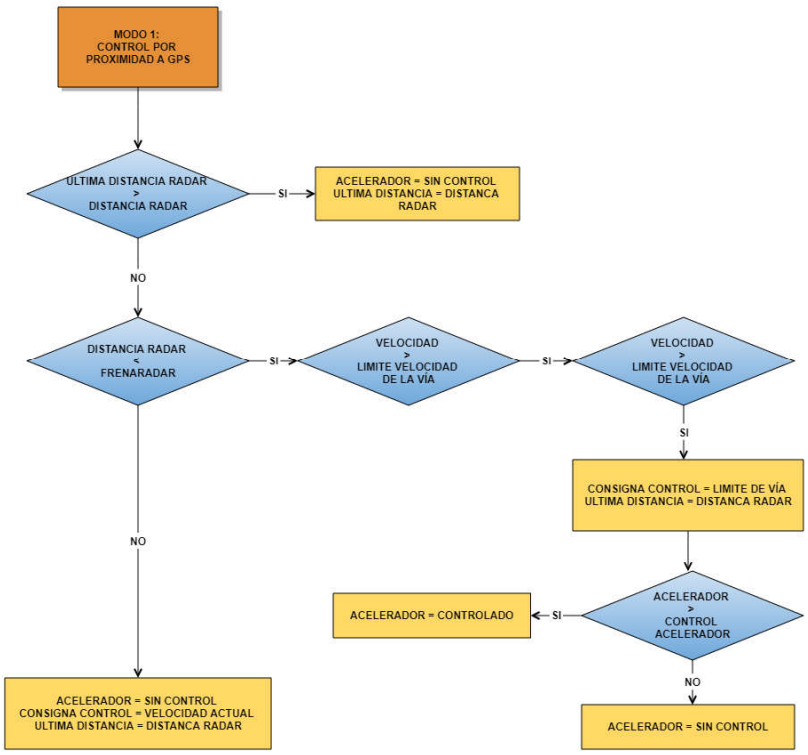


Ilustración 71: Diagrama de Modo 1. Control por proximidad a GPS

Como se puede ver el modo tiene diferentes funciones con un orden de prioridaddado por la instrucción condicional If-Else.

En el siguiente fragmento del código se comentan las partes más importantes, se puede consultar el código completo en el *Anexo II: Código de Arduino.*

- El control de velocidad se desactivará en caso de alejarse del punto GPS más cercano.

```
   if (distancia_radar > ult_distancia_radar){ //Función en caso de
alejarse del punto de radar fijo, dejar control automatico.
    ult_distancia_radar = distancia_radar; //Se actualiza ultima
señal de radar
    acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
    Setpoint = vel; //El controlador sigue funcionado para evitar
perturbaciones, con consigna igual a la velocidad real.
    }
```

- El control de velocidad estará activo en caso de una aproximación al punto de GPS para una distancia menor a la de calculada en “frenaradar” y se desactivará en el caso de que el vehículo vaya a una velocidad igual o menor a la velocidad límite de la vía. En el caso de que la acción controlada esté activada, predomina la acción de consigna real del acelerador, de esta forma el vehículo no mantendrá dicha velocidad en el caso de que el usuario deje de accionar e

acelerador, evitando posibles conflictos entre lo que el usuario intenta y el control automático.

En esta función se mantiene asignada la consigna de límite de velocidad de la vía aunque no esté activa la acción controlada, de esa forma se evitan perturbaciones en la acción de control, y la activación y desactivación del sistema automático es inapreciable, consiguiendo así una respuesta estable en todo momento.

```
   elseif ( (distancia_radar < frenaradar) and (vel > lim_vel) ){
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = lim_vel; //Consigna de limite de velocidad
referida a la velocidad de la via
        ult_distancia_radar = distancia_radar; //Se actualiza
ultima señal de radar
        }
      else { //Desconexión de sistemas automaticos
        acelerador_out = acelerador_in; //Señal de clonación igual
a señal real de acelerador.
        Setpoint = lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
        ult_distancia_radar = distancia_radar; //Se actualiza
ultima señal de radar
        }
    }
```

- Cuando ninguna de las acciones anteriores se cumplen, la acción de control será la aportada por la acción real del acelerador, obteniendo así una acción directa sin control automático. Al igual que en la función anterior, se mantiene asignada la consigna de límite de velocidad de la vía aunque no esté activa la acción controlada, de esa forma se evitan perturbaciones en la acción de control.

```
   else {
    ult_distancia_radar = distancia_radar;  //Se actualiza ultima
señal de radar
    acelerador_out = acelerador_in;  //Señal de clonación igual a
señal real de acelerador.
    Setpoint = vel; //El controlador sigue funcionado para evitar
perturbaciones, con consigna igual a la velocidad real.
    }
```

7.8.2.2 Modo 2: Limitador de velocidad por GPS y limitador de velocidad por consigna

La función de este modo trata de adecuar la velocidad del vehículo en el momento que éste se aproxime a un radar fijo o cualquier punto que se quiera añadir en el programa y la limitación de velocidad cuando lo anterior no ocurra.

Como la desaceleración no es la misma en todo el rango de velocidades a controlar, en este modo se compara la distancia al punto más cercano, con el resultado de multiplicar la velocidad por un factor constante (frenaradar = vel * K_frenaradar), de esta forma el vehículo no regulará su velocidad demasiado tarde en velocidades altas o demasiado pronto en la velocidades bajas.

Para la limitación de velocidad de forma manual, se programa en la función el incremento y decremento de una consigna de límite de velocidad con los botones del mando “V+” y ”V-”,

En la siguiente imagen se puede ver la estructura del modo 2:

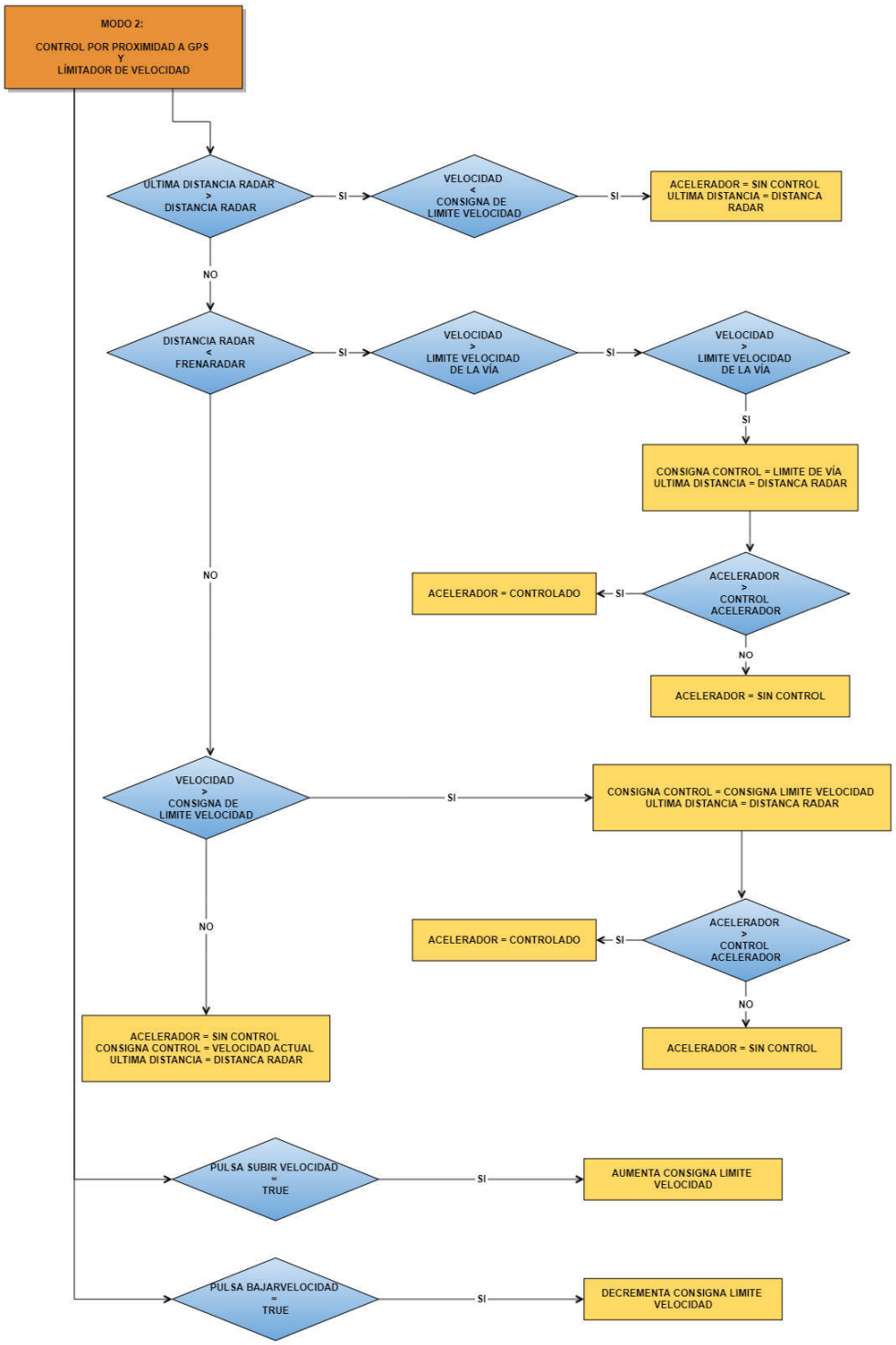


Ilustración 72: Diagrama de Modo 2. Control por proximidad a GPS y limitador de velocidad.

Como se puede ver el modo 2 tiene diferentes funciones con un orden de prioridad dado por la instrucción condicional If-Else. En el siguiente fragmento del código se

comentan las partes más importantes, se puede consultar el código completo en el *Anexo II: Código de Arduino.*

- En este modo se habilitan los pulsadores V+ y V- para el aumento o decremento de la consigna límite de velocidad.

```
if (P_VS ==true){  //Guarda consigna, incrementa o decrementa
   set_lim_vel = set_lim_vel +1;}
if (P_VB ==true){
   set_lim_vel = set_lim_vel -1;}
```

- El control de velocidad se desactivará en caso de alejarse del punto GPS más cercano y que el vehículo tenga una velocidad menor a la consigna de límite de velocidad.

```
   if ((distancia_radar > ult_distancia_radar) and (vel <
set_lim_vel)){ //Función en caso de alejarse del punto de radar
fijo y que la velocidad sea menor a la consigna limite de
velocidad, desactivar control
    ult_distancia_radar = distancia_radar; //Se actualiza ultima
señal de radar
    acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
    Setpoint = set_lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
    }
```

- El control de velocidad estará activo en caso de una aproximación al punto de GPS para una distancia menor a la de calculada en "frenaradar" y se desactivara en el caso de que el vehículo vaya a una velocidad igual o menor a la velocidad límite de la vía. En el caso de que la acción controlada esté activada, predomina la acción de consigna real del acelerador, de esta forma el vehículo no mantendrá dicha velocidad en el caso de que el usuario deje de accionar e acelerador, evitando posibles conflictos entre lo que el usuario intenta y el control automático.

```
   elseif ((vel >= lim_vel) and (distancia_radar < frenaradar)){
//Función para para el paso por radar fijo por GPS
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = lim_vel; //Consigna de limite de velocidad
referida al limite seleccionado
        ult_distancia_radar = distancia_radar;
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual
a señal real de acelerador.
        Setpoint = lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
        ult_distancia_radar = distancia_radar;
        }
    }
```

- Cuando la velocidad del vehículo llega o supera a la consigna de límite de velocidad se habilita la acción de control con dicha consigna, condicionada dicha acción de control a que la acción del acelerador real sea mayor a la del lazo de control.

```
   elseif ( vel > set_lim_vel ){ //Función para el control con
limitador de velocidad.
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = set_lim_vel; //Consigna de limite de velocidad
referida a la velocidad de la via
        ult_distancia_radar = distancia_radar;
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual
a señal real de acelerador.
        Setpoint = set_lim_vel; //El controlador sigue funcionado
para evitar perturbaciones
        ult_distancia_radar = distancia_radar;
        }
    }
```

- Cuando ninguna de las acciones anteriores se cumplen, la acción de control será la aportada por la acción real del acelerador, obteniendo así una acción directa sin control automático.

```
   else {
    ult_distancia_radar = distancia_radar;
    acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
    Setpoint = set_lim_vel;  //En modo reposo consigna límite, así
el control esta estable para el uso del control automatico
    }
```

7.8.2.3 Modo 3: Limitador de velocidad por GPS y control de velocidad (control de crucero)

La función de este modo se trata de adecuar la velocidad del vehículo en el momento que este se aproxime a un radar fijo o cualquier punto que se quiera añadir en el programa y el control de velocidad por consigna o control de crucero.

Como la desaceleración no es la misma en todo el rango de velocidades a controlar, en este modo se compara la distancia al punto más cercano, con el resultado de multiplicar la velocidad por un factor constante (frenaradar = vel * K_frenaradar), de esta forma el vehículo no regulará su velocidad demasiado tarde en velocidades altas o demasiado pronto en la velocidades bajas.

Para control de velocidad de forma manual, se programa en la función el incremento y decremento de una consigna de velocidad con los botones del mando “V+” y ”V-”.

En la siguiente imagen se puede ver la estructura del modo 3:

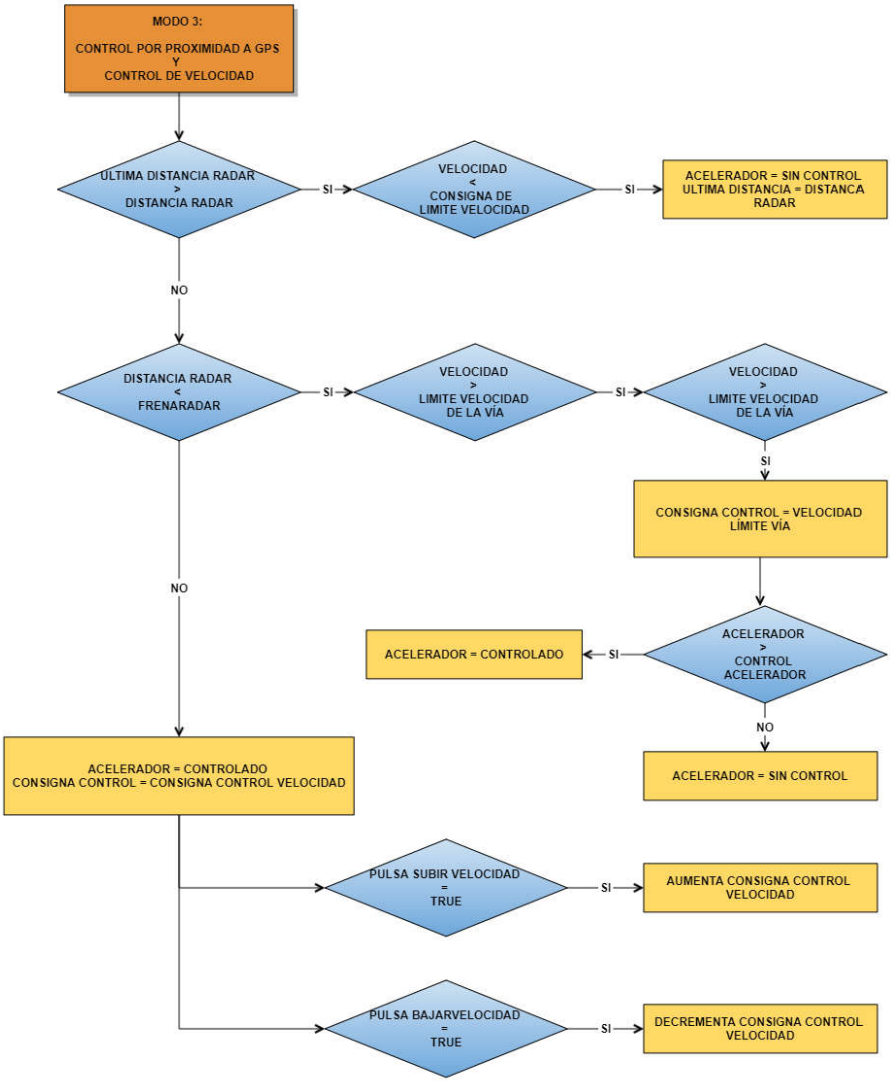


Ilustración 73: Diagrama de Modo 3. Control por proximidad a GPS y control de velocidad por consigna.

Como se puede ver el modo 3 tiene diferentes funciones con un orden de prioridad dado por la instrucción condicional If-Else. En el siguiente fragmento del código se comentan las partes más importantes, se puede consultar el código completo en el *Anexo II: Código de Arduino.*

- El control de velocidad por proximidad a GPS, se desactivará en caso de alejarse del punto GPS más cercano.

```
   if (distancia_radar > ult_distancia_radar){ //Función en caso de
alejarse del punto de radar fijo, dejar control por proximidad
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        ult_distancia_radar = distancia_radar;
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual
a señal real de acelerador.
        ult_distancia_radar = distancia_radar;
        }
    }
```

- El control de velocidad estará activo en caso de una aproximación al punto de GPS para una distancia menor a la de calculada en "frenaradar" y se desactivara en el caso de que el vehículo vaya a una velocidad igual o menor a la velocidad límite de la vía. En el caso de que la acción controlada esté activada, predomina la acción de consigna real del acelerador, de esta forma el vehículo no mantendrá dicha velocidad en el caso de que el usuario deje de accionar e acelerador, evitando posibles conflictos entre lo que el usuario intenta y el control automático.

```
   elseif ( (distancia_radar < frenaradar) and (vel > lim_vel) ){
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = lim_vel; //Consigna de limite de velocidad
referida a la velocidad de la via
        ult_distancia_radar = distancia_radar; //Se actualiza
ultima señal de radar
        }
      else { //Desconexión de sistemas automaticos
        acelerador_out = acelerador_in; //Señal de clonación igual
a señal real de acelerador.
        Setpoint = lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
        ult_distancia_radar = distancia_radar; //Se actualiza
ultima señal de radar
        }
    }
```

- Cuando ninguna de las acciones anteriores se cumplen, la acción de control será la aportada por la acción del lazo cerrado de control, activándose así el control de crucero. La velocidad de crucero será la consigna de límite de velocidad asignada justo antes, debiendo así de aumentar de forma manual la consigna de velocidad si se desea. Esta condición evitará que el vehículo se ponga a acelerar en el momento que se ha pasado un punto GPS de referencia sin que el usuario se lo espere.

```
   else {
```

```
    ult_distancia_radar = distancia_radar;  //Se actualiza ultima
señal de radar
    acelerador_out = acelerador_in;  //Señal de clonación igual a
señal real de acelerador.
    Setpoint = vel; //El controlador sigue funcionado para evitar
perturbaciones, con consigna igual a la velocidad real.
    }
```

7.8.2.4 Modo 4: Control de velocidad (control de crucero)

La función de este modo se trata de controlar la velocidad del vehículo por consigna o como se conoce de forma común, control de crucero.

Para control de velocidad de forma manual, se programa en la función el incremento y decremento de una consigna de velocidad con los botones del mando “V+” y ”V-”.

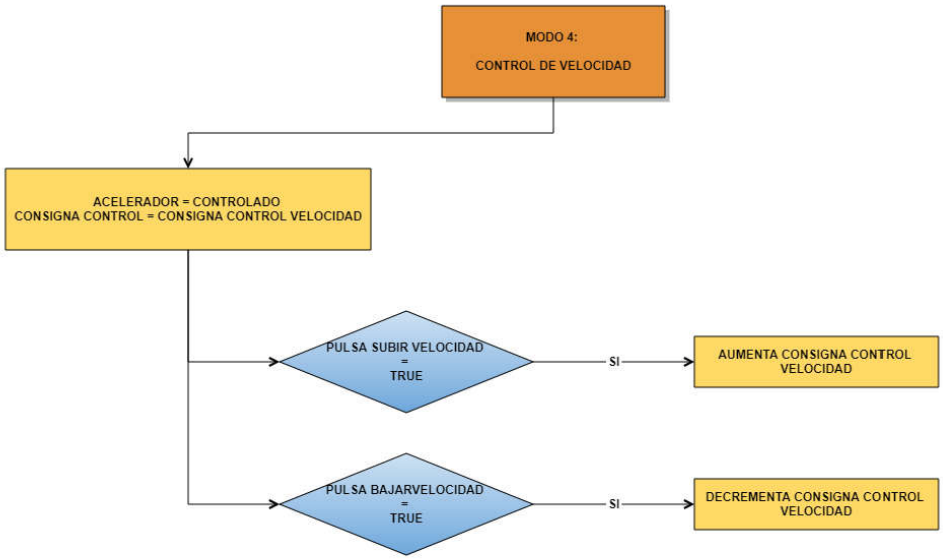


Ilustración 74:Diagrama de Modo 3. Control de velocidad por consigna.

En esta función no existen funciones condicionadas unas a otras, únicamente se activa la salida del acelerador clonada y controlada por el lazo cerrado de control. En el siguiente fragmento del código se comentan las partes más importantes de la función, se puede consultar el código completo en el *Anexo II: Código de Arduino*. Se memoriza el modo actual y se asigna la salida del acelerador la señal controlada. Al igual que el modo 3 programa la función para la configuración de la consigna deseada.

```
modo= 4;
acelerador_out = Outputint;
```

```
if (P_VS ==true){  //Guarda consigna, incrementa o decrementa
  Setpoint = Setpoint +1;}
if (P_VB ==true){
  Setpoint = Setpoint - 1;}
```

7.8.2.5 Modo 5: Limitador de velocidad por consigna

La función de este modo se trata de realizar la limitación de velocidad cuando se intente superar la consigna ajustada de límite de velocidad.

Para la limitación de velocidad de forma manual, se programa en la función el incremento y decremento de una consigna de límite de velocidad con los botones del mando “V+” y ”V-”,

En la siguiente imagen se puede ver la estructura del modo 5:

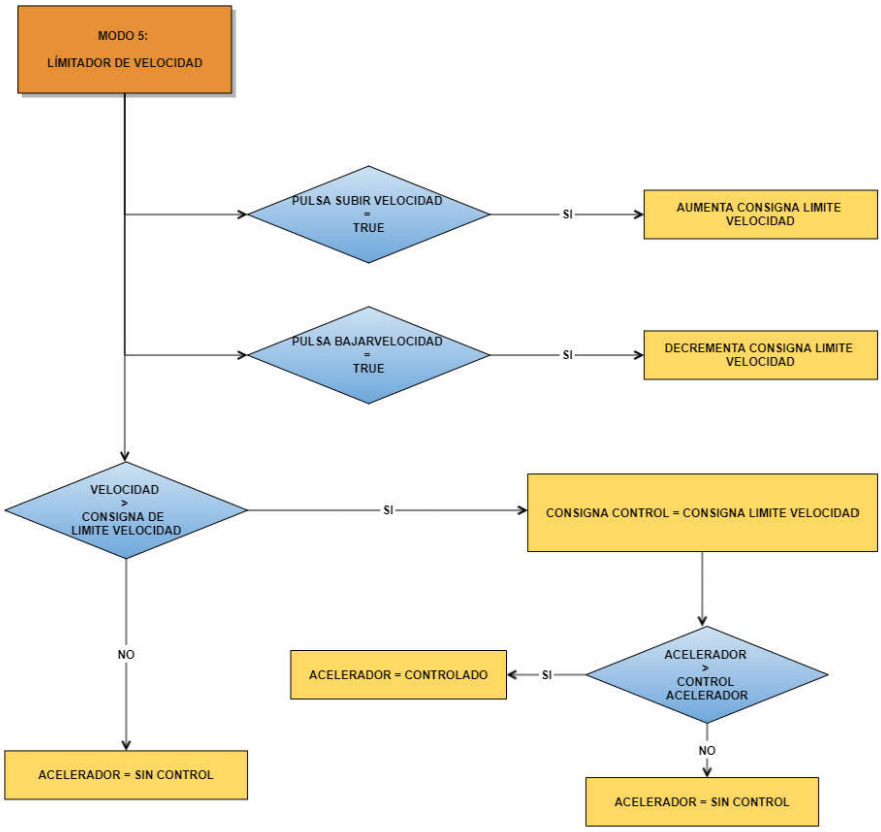


Ilustración 75:Diagrama de Modo 5. Limitador de velocidad por consigna.

El modo 5 tiene diferentes funciones con un orden de prioridad dado por la instrucción condicional If-Else. En el siguiente fragmento del código se comentan las partes más importantes, se puede consultar el código completo en el *Anexo II: Código de Arduino.*

- En este modo se habilitan los pulsadores V+ y V- para el aumento o decremento de la consigna límite de velocidad.

```
if (P_VS ==true){  //Guarda consigna, incrementa o decrementa
   set_lim_vel = set_lim_vel +1;}
if (P_VB ==true){
   set_lim_vel = set_lim_vel -1;}
```

- Cuando la velocidad del vehículo llega o supera a la consigna de límite de velocidad se habilita la acción de control con dicha consigna, condicionada dicha acción de control para que la acción del acelerador real sea mayor a la del lazo de control. En esta función se mantiene asignada la consigna de límite de velocidad aunque no esté activa la acción controlada, de esa forma se evitan perturbaciones en la acción de control, y la activación y desactivación del sistema automático es inapreciable, consiguiendo así una respuesta estable en todo momento.

```
   elseif ( vel > set_lim_vel ){ //Función para le control con
limitador de velocidad.
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = set_lim_vel; //Consigna de limite de velocidad
referida a la velocidad de la via
        ult_distancia_radar = distancia_radar;
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual
a señal real de acelerador.
        Setpoint = set_lim_vel; //El controlador sigue funcionado
para evitar perturbaciones
        ult_distancia_radar = distancia_radar;
        }
    }
```

- Cuando ninguna de las acciones anteriores se cumplen, la acción de control será la aportada por la acción real del acelerador, obteniendo así una acción directa sin control automático. Al igual que en la función anterior, esta función mantiene asignada la consigna de límite de velocidad aunque no esté activa la acción controlada, de esa forma se evitan perturbaciones en la acción de control.

```
   else {
    ult_distancia_radar = distancia_radar;
    acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
```

```
    Setpoint = set_lim_vel;  //En modo reposo consigna límite, así
el control esta estable para el uso del control automatico
```

7.8.3 Programa de la función mando ()

Como se puede consultar en este documento, en el apartado “Prueba de mando radio cd” se parte de una señal analógica, cuyos valores están relacionados a la pulsación de los botones del mando. Con ayuda de los datos recogidos para la utilización de este mando se programa la función mando(), en la que existen dos funciones principales: la función de anti-rebote, para eliminar perturbaciones del los botones y la asignación de una memoria a la selección de dicho botón. A continuación se comentan las partes más importantes del código, se puede consultar el código completo en el *Anexo II: Código de Arduino.*

En las siguientes funciones se compara el dato de entrada, y dependiendo de la prioridad antes comentada se aumenta el valor de la variable correspondiente por ciclo de Scan, haciendo así la función de anti-rebote con filtro digital. Se programan funciones de tres formas diferentes para obtener tres acciones distintas de salida con la misma acción de entrada, todas ellas con el filtro digital para el anti-rebote:

- Para habilitar dos memorias para un mismo pulsador, una para la detección de pulsación corta y la otra para pulsación larga.
- Para habilitar una memoria cuando pasa un intervalo de tiempo durante un único ciclo de programa. Esta configuración la tienen los pulsadores Seek Up y Seek Down.
- Para habilitar una memoria cuando pasa un intervalo de tiempo durante un único ciclo de programa, de forma cíclica hasta dejar de pulsar. Esta configuración la tienen los pulsadores V+ y V-.

```
void mando() {

 valor_mando =analogRead(analogInPin6);

if (valor_mando < 100) { // Analiza boton pulsado y antirrebote
errormando= 1;} //Error por mando desconectado
elseif (valor_mando < 300) { //Mando conectado
errormando= 0;
cuentaVS= 0;
cuentaVB= 0;
cuentaSS= 0;
cuentaSB= 0;
cuentamode= 0;
P_mode_L =false;
P_mode_C =false;
P_VS =false;
P_VB =false;
```

```
P_SS =false;
P_SB =false;}
elseif (valor_mando < 570) { // Pulsador mode
  cuentamode= cuentamode+1;}
elseif (valor_mando < 700) { //Pulsador Seek subir
  cuentaSS= cuentaSS+1;
  P_SS =false;}
elseif (valor_mando < 820) { //Pulsador Seek bajar
  cuentaSB= cuentaSB+1;
  P_SB =false;}
elseif (valor_mando < 930) { //Pulsador Velocidad subir
  cuentaVS= cuentaVS+1;
  P_VS =false;}              //Condicion para generar pulsos con
entrada activa
elseif (valor_mando < 1020) { //Pulsador Velocidad bajar
  cuentaVB= cuentaVB+1;      //Condicion para generar pulsos con
entrada activa
  P_VB =false;}

if (cuentamode > 200){//Valida pulsacion
P_mode_C =false;
P_mode_L =true;}
if (cuentamode == 10)//Valida pulsacion
P_mode_C =true;
if (cuentaVS > 60){//Valida pulsacion
P_VS =true;
cuentaVS= 0;}
if (cuentaVB > 60){//Valida pulsacion
P_VB =true;
cuentaVB= 0;}
if (cuentaSS == 200)//Valida pulsación para un único ciclo de
programa.
P_SS =true;
if (cuentaSB == 200)//Valida pulsación para un único ciclo de
programa.
P_SB =true;
}
```

- En la siguiente función se compara el dato de cada una de las variables de cuenta y activa el valor del botón correspondiente, las funciones están condicionadas por los tipos de acción que se acaban de detallar en la función anterior.

7.8.4 Programa de la función vel()

Con ayuda de los datos recogidos para la lectura de la velocidad se programa la función vel(). Como se puede consultar en este documento, en el apartado “Prueba de lectura de velocidad” se utilizan unas ecuaciones por tramos, cuyos valores están relacionados a la velocidad. A continuación se detallan las partes importantes del código, auque se dispone del programa cmpleto en el *Anexo II: Código de Arduino.*

- Lectura de las entradas analógicas

```
Bobina11 =analogRead(analogInPin0);
Bobina12 =analogRead(analogInPin1);
Bobina21 =analogRead(analogInPin2);
Bobina22 =analogRead(analogInPin3);
```

- Función para el cálculo de la velocidad, a través de las comparaciones de los datos de entrada analógicos para la selección de los tramos relacionados con las ecuaciones correspondientes.

```
if ( Bobina21 > Bobina22) {
  vel= (-0.0539*Bobina21 + 65);
} elseif( Bobina22> Bobina21) {
      if ( Bobina22 > Bobina12) {
            if (Bobina11 > Bobina12) {
              vel= (0.0571*Bobina11 + 130);
            } else {
              vel= (-0.0543*Bobina12 + 142);
                              }
    } else {
      vel= ( 0.0557 * Bobina22 + 53);
    }

} else {
  vel= 200;
  }
```

7.8.5 Programa de la función acelera()

A partir de los datos y la función programada en la pruebas, como se puede consultar en este documento, en el apartado “Prueba de lectura de acelerador”, para la clonación de la señal son necesarias tres señales de entrada, que se comparan después de un escalado de 0-100 % para comprobar que el funcionamiento es correcto, y tres señales de salida. Estas señales de salida se escalan con los datos conocidos obteniendo así una acción de aceleración controlada o no. Para controlar la acción de control se asignaran valores reales de la lectura del acelerador (“acelerador_in”) o valores resultantes de la acción de control en lazo cerrado (“ouputint”) al dato a escalar en las salidas (“acelerador_out”). A continuación se detallan las partes importantes del código, auque se dispone del programa cmpleto en el *Anexo II: Código de Arduino.*

- Lectura de las entradas analógicas

```
acelerador_in =map(analogRead(acel_in_4), 220, 750, 0, 100); //
Conexión a pin 4 del acelerador, escalado %
acelerador_in_5 =map(analogRead(acel_in_5), 220, 750, 0, 100); //
Conexión a pin 5 del acelerador, escalado %
```

```
acelerador_in_7 =map(analogRead(acel_in_4), 220, 750, 0, 100); //
Conexión a pin 7 del acelerador, escalado %
```

- Comprueba que los datos de entrada son parecidos después de ser escalados, en caso contrario se activa una memoria de error de acelerador.

```
if ((((acelerador_in + acelerador_in_5 + acelerador_in_7)/3)-
acelerador_in) > 10)
error_acelerador =true;

if ((((acelerador_in + acelerador_in_5 + acelerador_in_7)/3)-
acelerador_in) > 10)
error_acelerador =true;
```

- Escalador de las salidas analógicas con el valor asignado a “acelerador_out”.

```
  outputValue2 =map(acelerador_out, 0, 100, 66, 199);
//65.79=1.29*255/5  pin 4
  outputValue3 =map(acelerador_out, 0, 100,221,43 ); // pin 5
  outputValue4 =map(acelerador_out, 0, 100, 36, 171); // pin 7
```

- Actualización de las salidas analógicas con el valor escalado.

```
 analogWrite(analogOutPin2, outputValue2);
  analogWrite(analogOutPin3, outputValue3);
  analogWrite(analogOutPin4, outputValue4);
}
```

7.8.6 Programa de la función bluetooht ()

Para la visualización de los datos a través del Smartphone se programa esta función que se encarga de comunicarse con el puerto Serial3 del Arduino. Está dividida en 3 partes significativas: espera a la recepción de nuevos datos, activación de señales dependiendo de los caracteres recibidos y envío de datos cuando la aplicación lo solicita. Esta misma función, la cual se encuentra de forma completa en el *Anexo II: Código de Arduino*, es la encargada de comunicarse con la App_v5 de este proyecto.

- Cada vez que se reciba un carácter del módulo Bluetooth se configura una acción. Es de esa forma como se van a dedicar determinados caracteres para diferentes acciones e el programa del microcontrolador.

```
void bluetooht(){
if(Serial3.available()>0){           // Si el puerto serie esta
habilitado
        char_blue =Serial3.read();       // Lee lo que llega por el
puerto Serie
```

```
}
```

- Cada vez que se reciba un carácter del módulo bluetooht se configura una acción. Es de esa forma se asigna un carácter concreto para cada función a realizar.

```
if(char_blue=='1'){                    // on/off de pin 136
      digitalWrite(13,HIGH);
 }
if(char_blue=='2'){                    // on/off de pin 13
      digitalWrite(13,LOW);
 }
```

- De esa misma forma, se asigna el carácter "R" para la solicitud del envío de datos por parte de la App del Smartphone, así como el carácter "|" le indica a la App que ha finalizado el dato enviado.

```
  if(char_blue=='R'){       // envia todos los datos cuando la
aplicación lo solicita (R)
  Serial3.print(estado);
  Serial3.print("|");
  Serial3.print(acelerador_out);
  Serial3.print("|");
  Serial3.print(vel);
  Serial3.print("|");
  Serial3.print(distancia_radar);
  Serial3.print("|");
  Serial3.print(error,2);
  Serial3.print("|");
  Serial3.print(ITerm,2);
  Serial3.print("|");
  }
 char_blue=0;
 }
```

7.8.7 Programa de control, función lazo_control ()

Para el control de velocidad se ha programado una función de control, la cual corresponde a un lazo de control cerrado con un controlador PI. Se ha llevado a cabo con este tipo de controlador principalmente por su sencillez. La elección de este tipo de controlador se debe a la seguridad, ya que se consigue un sistema lo bastante rápido para las necesidades del proyecto y lo suficientemente lento para asegurar que se controla el sistema de una forma estable.

Esto se debe a tener en cuenta que en el proyecto existen muchas variables que cambiarán dependiendo de cada situación; como son la resistencia del viento dependiente de la velocidad, las pendientes por las que circula el vehículo, la velocidad seleccionada en la caja de cambios.

Además de la selección del control PI por ser el método más seguro, en los ensayos reales para el ajuste se implementan dos funciones Anti-Windup, una para la acción de control y otra para la acción integral. En la siguiente imagen se puede observar el tipo de control programado.

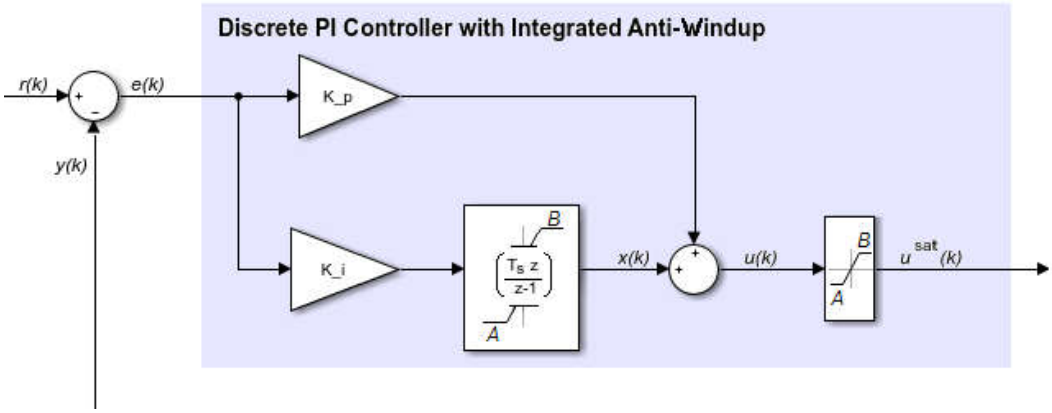


Ilustración 76: Control PI

Con ayuda de los datos obtenidos en la lectura de del velocímetro y de forma experimental se ajustan las ganancias del controlador kp y ki, hasta obtener la dinámica lo suficientemente lenta para que sea estable y segura. En la siguiente imagen se observa cómo se han hecho las pruebas, tomando datos en tiempo real de las acciones del controlador.

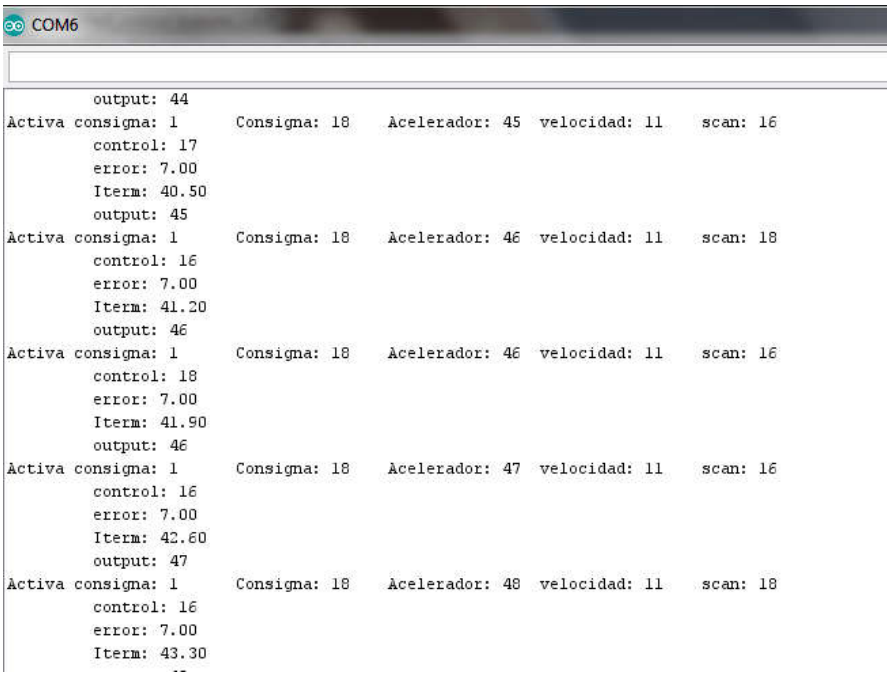


Ilustración 77: Pruebas de configuración del controlador PI

A continuación se detallan los fragmentos más importantes del código de la función lazo_control(). Esta misma función, la cual se encuentra de forma completa en el *Anexo II: Código de Arduino.*

- Para una ejecución de forma periódica se utiliza un valor guardado de la última ejecución y se compara con el tiempo total de ejecución del registro millis().

```
//Función para la ejecución por periodos de muestreo del control
t_control = (ultimo_scan - f_control);
if(t_control > t_muestreo_control)
```

- Cálculo del error.

```
error= Setpoint - vel;
```

- Cálculo del término integral, resultado del acumulado más el error actual.

```
ITerm = ITerm + (ki * error);
```

- Anti windup del error acumulado. Esto evitará que error acumulado se haga muy grande de forma innecesaria, ya que esto haría de un sistema inestable si el control automático se conectara en ese momento.

```
//Aunti windup del error acumulado
if (ITerm > 20){
  ITerm = 20; }
if (ITerm < (-20){
  ITerm = 20; }
```

- Cálculo de la acción de control, del término proporcional y sumatorio de término proporcional e integral. Se convierte el resultado a un dato de tipo entero, para su uso en el escalado de las señales analógicas.

```
// Calculamos la función de salida del PI.
Output = kp * error + ITerm;
Outputint = Output; // convierte el decimal a entero
actcontrol=true; //Refistro de activación dela función control. Bit
para las acciones temporizadas (serial PC)
```

- Actualiza la memoria con el último tiempo registrado para su comparación en sucesivos ciclos de programa.

```
f_control = ultimo_scan;
```

- Anti windup para la acción de control, evitando así acciones del controlador agresivas para el sistema.

```
if (Outputint > 75){ //Aunti windup
Outputint =75;}
if (Outputint < 4){ //Aunti windup
Outputint =4;}
```

7.8.8 Programa para la selección de radares próximos, función lista_gps()

Para el control de velocidad por proximidad a un radar fijo de los que se encuentran publicados por la DGT, es necesario hacer una selección de aquellos puntos se que encentren próximos cuando el vehículo este circulando. Se han seleccionado únicamente aquellos que se encuentran en las proximidades de la realización de las pruebas de funcionamiento, así como puntos imaginarios igualmente validos para dichas pruebas.

En la siguiente imagen se muestra un ejemplo del código HTML que proporciona la DGT de los radares fijos publicados en su Web, de esta lista se recogen los datos de Latitud, Longitud y sentido de detección del radar:

```
▼<CARRETERA>
   <DENOMINACION>CM-3215</DENOMINACION>
  ▼<RADAR>
   ▼<PUNTO_INICIAL>
      <PK>0.61</PK>
      <LATITUD>38.61497</LATITUD>
      <LONGITUD>-1.50566</LONGITUD>
    </PUNTO_INICIAL>
   ▼<PUNTO_FINAL>
      <PK>17.09</PK>
      <LATITUD>38.60391</LATITUD>
      <LONGITUD>-1.67337</LONGITUD>
    </PUNTO_FINAL>
    <SENTIDO>Ambos</SENTIDO>
  </RADAR>
```

Ilustración 78: Radares fijos por código HTML

Se podría extraer de esta lista los radares e insertarlos en un formato válido para la introducción en el microcontrolador, aunque en este caso se decide realizar una lista solo de aquellos puntos que sean necesarios para las pruebas en las proximidades. Se eligen los puntos necesarios y se ordenan según su Latitud, de esta forma no será necesario comparar todos ellos, reduciendo significativamente los recursos de cálculo del microcontrolador.

Desde una lista de las que se pueden encontrar en la red o con la extracción de los datos en HTML, se obtiene un documento de texto sin formato.

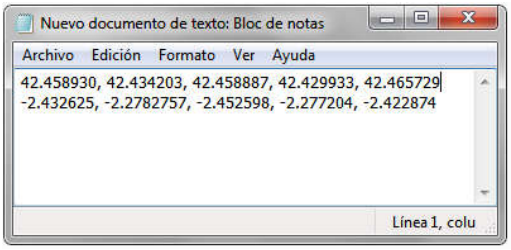


Ilustración 79: Lista de radares .txt

Esta lista la ordenaremos en función de la latitud o la longitud con ayuda de un software adecuado, en este caso se ha usado Excel.

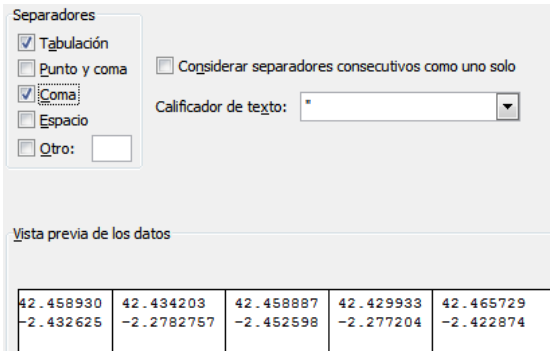


Ilustración 80: Conversión de datos

Obteniendo así la posibilidad de ordenar los datos en función del código que se emplea en el microcontrolador, como se muestra en el siguiente resultado.

	A	B	C	D	E
1	42.465729	42.458930	42.458887	42.434203	42.429933
2	-2.422874	-2.432625	-2.452598	-2.2782757	-2.277204

Ilustración 81: Datos ordenados en función de la Latitud

A continuación se detallan los fragmentos más importantes del código de la función lista_gps(). Esta misma función, la cual se encuentra de forma completa en el *Anexo II: Código de Arduino.*

- Se insertan los datos ordenados en dos vectores, los cuales se han ordenado para el uso en esta función. De la misma forma se inserta otro vector para el registro de la dirección detección, aunque finalmente no se ha puesto en práctica en este proyecto, solo sería necesario implementarlo en la máquina de estados o en la función "gps()" antes de realizar los cálculos.

```
- // Radar {Lobete, Sequero, Sequero2, Duques, Universidad}
- float ref_LAT[] = { 42.465729, 42.458930, 42.458887, 42.434203,
  42.429933};
- float ref_LON[] ={ -2.422874,-2.432625,-2.452598,-2.2782757,-
  2.277204};
- float ref_direccion[]={0, 0, 0, 0, 0}; //Orientación en la que
  detecta el radar, 0 para ambos sentidos
```

- En la función se comparan los datos de proximidad al punto de referencia anterior con el punto siguiente y el punto anterior de la lista. De esta forma se cambiara el punto de referencia más próximo en caso de que existan cambios.

```
void lista_gps()
```

```
{

dif_LAT_ultima = flat - ref_LAT[p];//Guarda la diferencia con
puntero ultima
dif_LON_ultima = flon - ref_LON[p];//Guarda la diferencia con
puntero ultima

if((flat - ref_LAT[p++]) < dif_LAT_ultima){
p = p++; //Incrementar el puntero para la coordenada mas proxima
}
if((flon - ref_LAT[p--]) < dif_LAT_ultima){
p = p--; //Decrementar el puntero para la coordenada mas proxima
}

dif_LAT_ultima = flat - ref_LAT[p];//Guarda la diferencia con
puntero actual
dif_LON_ultima = flon - ref_LON[p];//Guarda la diferencia con
puntero actual
```

7.8.9 Programa para el cálculo de la distancia a radares próximos. Código de la función gps ()

Para el posicionamiento por coordenadas GPS y el cálculo de de las distancias a los puntos a los que se adecuará la velocidad del vehículo en el momento que este se aproxime a un radar fijo o cualquier punto que se quiera añadir en el programa, como pueden ser cruces peligrosos o puntos de alta siniestralidad, se programa la función gps(). En esta función se diferencian tres partes significativas: adquisición de datos a través del puerto serie2 para la comunicación con el módulo GPS, selección del punto más cercano en la lista y cálculo de la distancia a dicho punto.

Para la adquisición de datos se utiliza una librería que facilita mucho la programación, llamando a instrucciones de esta librería se recogen los siguientes datos:

- La longitud y latitud, para el cálculo de proximidad a un radar fijo.
- El estado OK del receptor para saber si los datos están disponibles.
- La velocidad para la calibración más precisa de la lectura del indicador de velocidad si fuera necesario.
- La dirección en la que se está moviendo el vehículo para comparar la dirección en la que detecta el radar fijo.
- La variable “Sentences”, para saber si existen nuevos datos o se repiten, por falta de señal o mala señal GPS.

Para el cálculo de las distancias a través de las coordenadas GPS existen muchos modelos matemáticos para la aproximación de la tierra a un elipsoide o a una esfera, como son los modelos como el algoritmo iterativo conocido como fórmulas de Vincenty que proporciona una precisión extremadamente buena. El problema de este método es que al ser iterativo, puede provocar un bucle infinito. Existen otros

métodos para el cálculo de la distancia entre dos puntos situados en la superficie de una esfera, como la fórmula de Haversine, que es la más precisa frente a otras alternativas como la ley de los cosenos o el teorema de Pitágoras. En el caso de las necesidades de este proyecto se prioriza el bajo consumo de recursos que ofrece la ley de los cosenos y el teorema de Pitágoras a la precisión, ya que no es necesaria, pudiendo asumir errores muy grandes con las necesidades del proyecto.

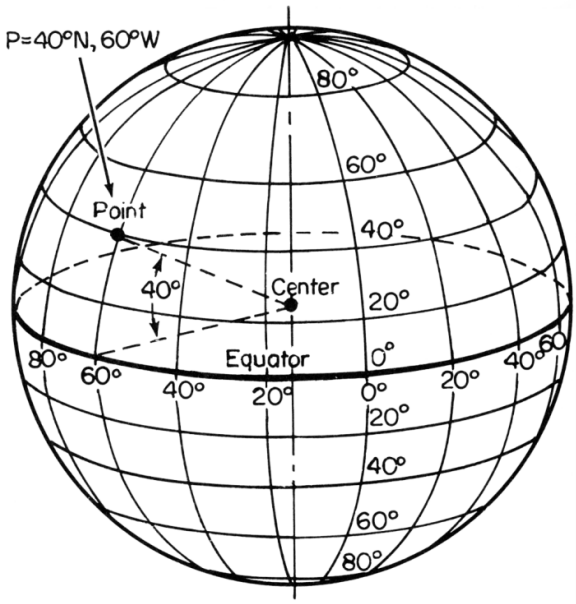


Ilustración 82: Latitud y longitud en el modelo de la esfera

En distancias pequeñas la diferencia entre el ángulo real y el ángulo de referencia tiende a cero, y por los tanto se puede aproximar el arco a una línea recta, ya que cuanto más cerca esté el vehículo del punto de referencia, menor será el error. Es de esta forma como se han calculado las distancias a el punto más cercano de control, tanto para los ángulos de latitud como de longitud.

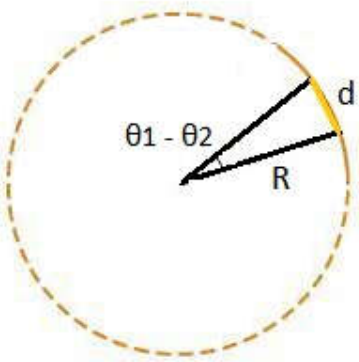


Ilustración 83: Aproximación del cálculo de distancias.

Los cálculos para implementar en el microcontrolador son los siguientes:

- Cálculo de la componente de Latitud.

$$f\,d_x = R \quad \sin\left((\text{flat} - \text{ref}_{LAT}) \quad \frac{PI}{180}\right)$$

- Cálculo de la componente de Longitud.

$$d_y = R \quad \sin\left((\text{flon} - \text{ref}_{LON}) \quad \frac{PI}{180}\right)$$

- Cálculo de la distancia resultante por Pitágoras

$$\text{distancia}_{radar} = \text{sqrt}(d_y \quad d_y + d_x \quad d_x)$$

A continuación se detallan los fragmentos más importantes del código de la función f_gps(). Esta misma función, la cual se encuentra de forma completa en el *Anexo II: Código de Arduino.*

- Recibe las tramas del módulo de GPS y comprueba si hay datos nuevos cada segundo.

```
  // Rrecibir secuencia durante un segundo
t_gps = (ultimo_scan - tf_gps);
if(t_gps>=t_muestreo_gps)
{
      while (Serial2.available())
      {
         char c =Serial2.read();
         if (gps.encode(c)){ // Nueva secuencia recibida
            newData=true;
            tf_gps =millis();}
      }
```

- En el caso de recibir datos nuevos se usa la instrucción TinyGPS para extraer los datos necesarios de la trama recibida.

```
if (newData){
  flat==TinyGPS::GPS_INVALID_F_ANGLE?0.0 : flat, 6;
  flon==TinyGPS::GPS_INVALID_F_ANGLE?0.0 : flon, 6;
  velgps= gps.f_speed_kmph();
  direccion= gps.f_course();
```

- Actualización del punto más próximo de la lista de coordenadas GPS.

```
//Actualización del punto de referencia más proximo
 lista_gps();
```

- Cálculo a la distancia de referencia.

```
 //Calculo de distancia al punto de referencia
d_x = R * sin((dif_LAT_ultima)* PI/180);
d_y = R * sin((dif_LON_ultima)* PI/180);
distancia_radar =sqrt(d_y*d_y+d_x*d_x);
```

7.8 Conclusiones

7.8.1 Objetivos alcanzados

Se ha alcanzado el objetivo de dotar al vehículo de asistencias electrónicas al control de velocidad. Todo ello con el uso de electrónica de bajo coste: la placa de desarrollo con microcontrolador Arduino MEGA, módulos NEO-6M para las de posicionamiento GPS y comunicación Bluetooth con el módulo HC-06. El proyecto finaliza con las siguientes soluciones:

- Lectura de la velocidad real del vehículo a través de la adquisición de las tensiones de las bobinas del indicador de velocidad.
- Para obtener el control del acelerador, pasa de ir conectado directamente a el módulo de control del vehículo a ir conectado a el microcontrolador Arduino. Se trata de tres señales analógicas con diferentes rangos de trabajo, las cuales ha sido necesario interpretar para los dos modos de trabajo actuales del acelerador: la clonación de las señales del acelerador y la asignación de valores por el lazo de control. Para ello se asignan tres pines PWM de Arduino, con filtro RC, que han sido conectados a la unidad de control del vehículo.
- Se instala el módulo NEO-6M para la adquisición de coordenadas GPS, necesaria para detección de radares próximos.
- Instalación del módulo Bluetooth y del diseño de la aplicación para el Smartphone para la visualización de datos en tiempo real. Esto dota al proyecto de una gran transparencia en funcionamiento, ya que se visualizan en el Smartphone los datos que se deseen del microcontrolador; velocidad, acción del acelerador, modo de funcionamiento, consigna de control, distancia del punto GPS más próximo.
- Adaptación de mando Radio-CD para la configuración del sistema. A través de una entrada analógica del microcontrolador, se analiza el botón pulsado del mando. Se obtiene de esta forma un mando situado en el volante y con botones suficientes para manipulación excelente de las funciones del sistema e control y de la asignación de consignas.

7.8.2 Problemas encontrados

Existen muchos problemas que han surgido a los largo del proyecto, algunos de los que ha dependido la viabilidad del mismo. El problema más significativo que ha repercutido a lo largo de todo el proyecto ha sido la falta de información de las características del vehículo. La documentación de taller es extensa y está bien detallada, pero se trata de documentaciones para el usuario, o un tipo de documentación para las reparaciones de taller.

Cuando se inicia el proyecto se parte de suposiciones, o análisis rápido de los componentes del vehículo, esto se ha traducido descubrir los problemas y las soluciones realizando análisis de las señales o comunicaciones.

En el caso de la adquisición de velocidad, fue imposible hacer la lectura de ningún valor en el bus CAN, aunque los niveles de tensión eran correctos y las conexiones que pertenecen al bus CAN estaban cableadas. También falló la idea de hacer la lectura de velocidad a través del sensor ABS, normalmente se trata se sensores magnéticos de los que se puede extraer la velocidad con un pequeño integrado, pero en este caso el sensor tampoco es de ese tipo. Finalmente hubo que optar por una forma “bruta” de realizar la medida de velocidad leyendo las tensiones del indicador de velocidad.

Para la clonación de señal se hizo una análisis completo, tanto de su funcionamiento en conectado como desconectado, ya que la información no era suficiente para empezar el diseño. No surgieron tantos problemas a la hora de la lectura de las señales clonadas por el módulo de control del vehículo, excepto que costo detectar que el módulo de control empezaba a funcionar antes que el microcontrolador Arduino, lo que hacía que el vehículo entrara en modo seguro.

7.8.3 Vías de continuación

Desde el inicio del proyecto existían otras ideas de mejora aprovechando los que está diseñado. Siguiendo con mejoras al sistema desarrollado sería interesante:

- La instalación de un medidor laser, para la regulación de velocidad manteniendo la distancia con el vehículo que circulara delante.
- Instalación de un módulo con tarjeta de memoria, para la actualización de los puntos GPS de radares.
- Automatización del encendido de las luces, sería algo sencillo y la documentación del vehículo sería suficiente.
- Automatización de la velocidad de los limpiaparabrisas, con regulación de velocidad manual o automática por cantidad de lluvia.
- Sensores de ultrasonidos para facilitar el aparcamiento.

Todas estas funciones adicionales se podrían añadir al proyecto existente, ya que se han consumido pocos recursos del Arduino MEGA.

UNIVERSIDAD DE LA RIOJA

ANEXO I: ESQUEMAS ELÉCTRICOS FORD FOCUS

Desarrollo de Sistema de Control de Velocidad para automóvil sincronizado con la base de datos de cinemómetros de la DGT

UNIVERSIDAD DE LA RIOJA
GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Firma del presente documento:

Iván Peña Moreno

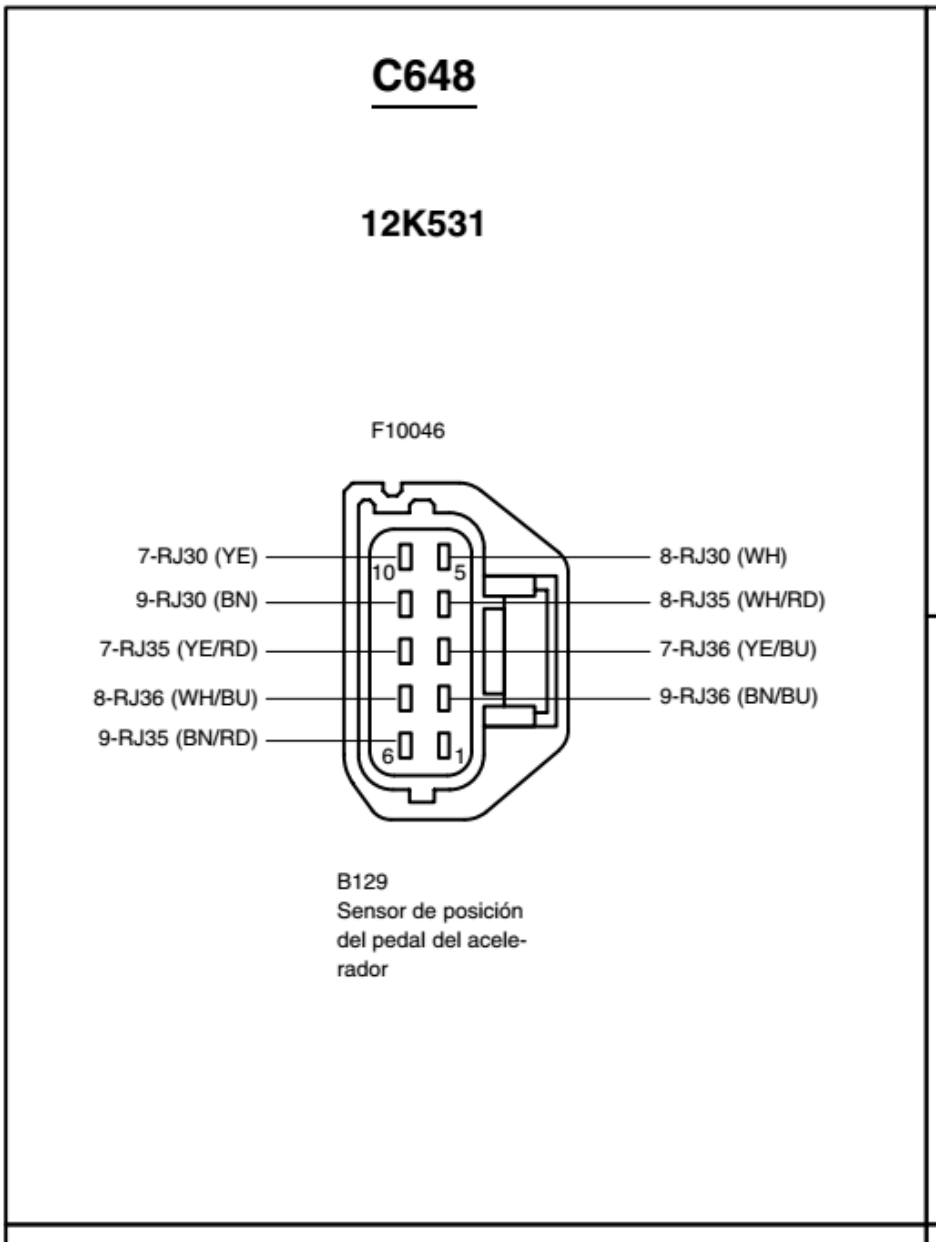
UNIVESIDAD DE LA RIOJA

Logroño, 3 de Marzo de 2019

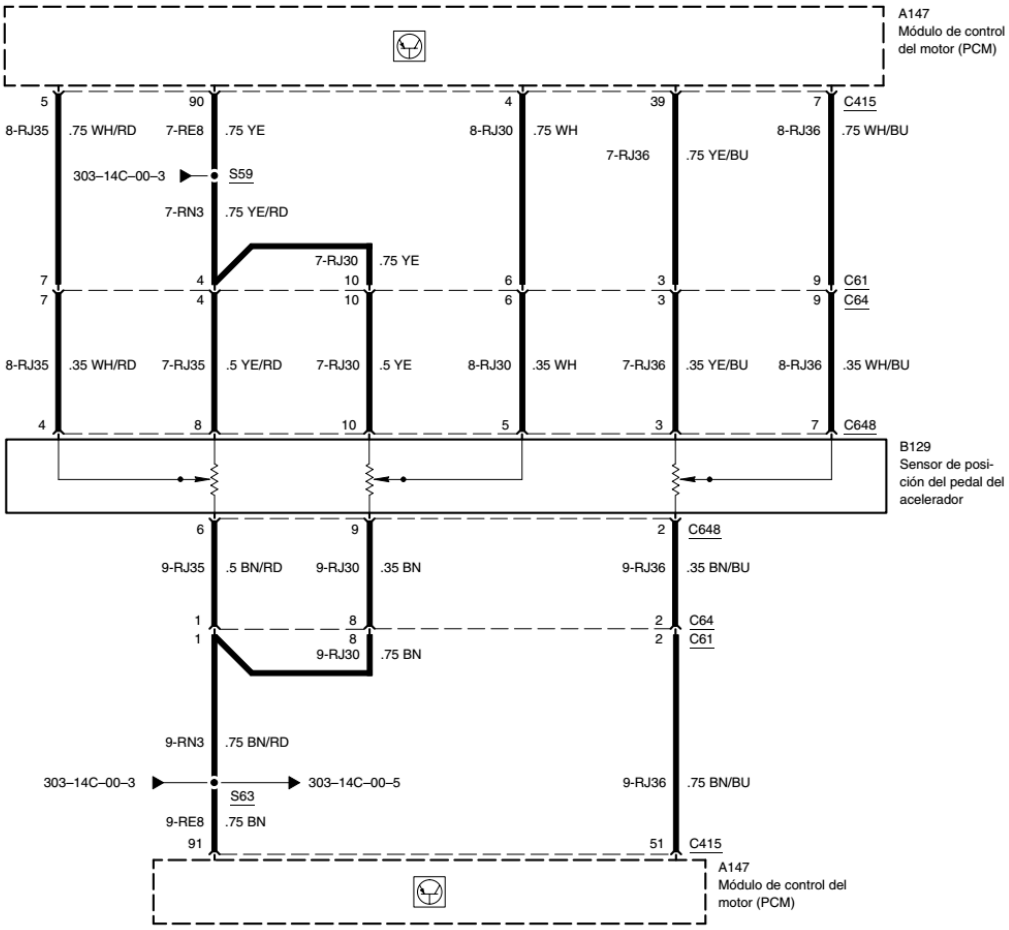
ANEXO I: ESQUEMAS ELÉCTRICOS FORD FOCUS

1. Documentación de esquemas eléctricos de Ford Focus 1.8TDCi DAW

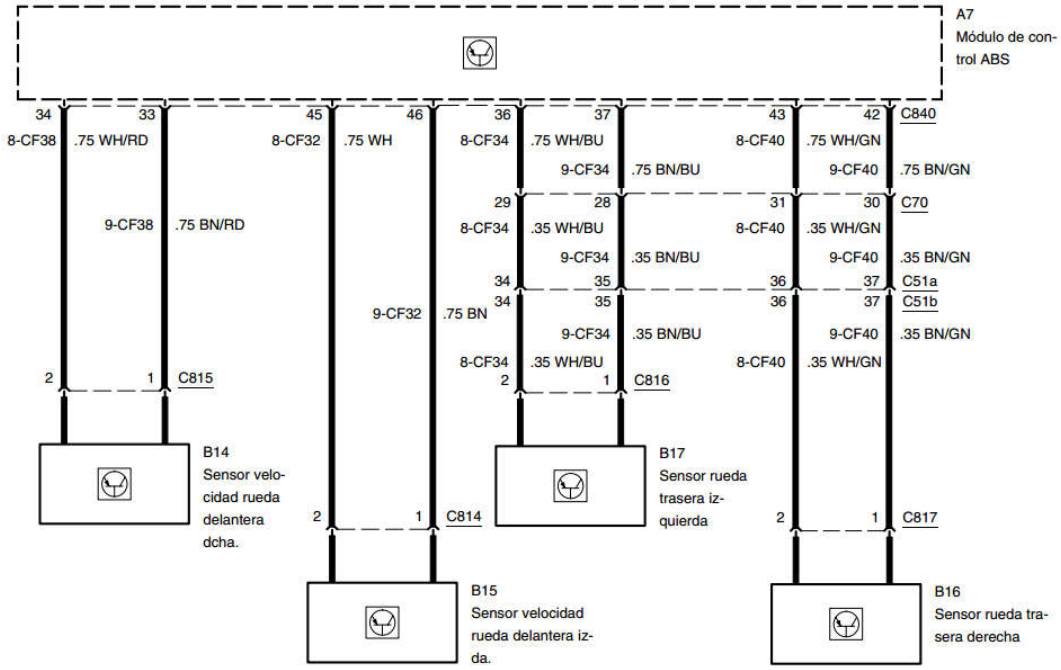
1.1 Conector de acelerador



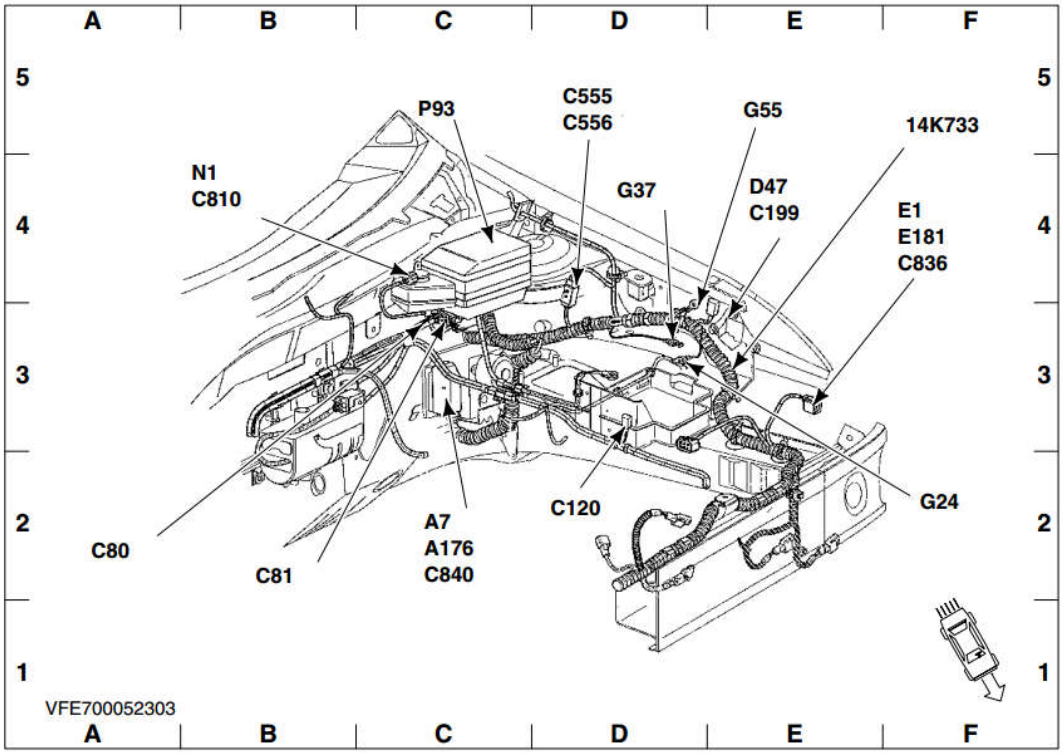
1.2 Esquema de conexión de pedal de acelerado



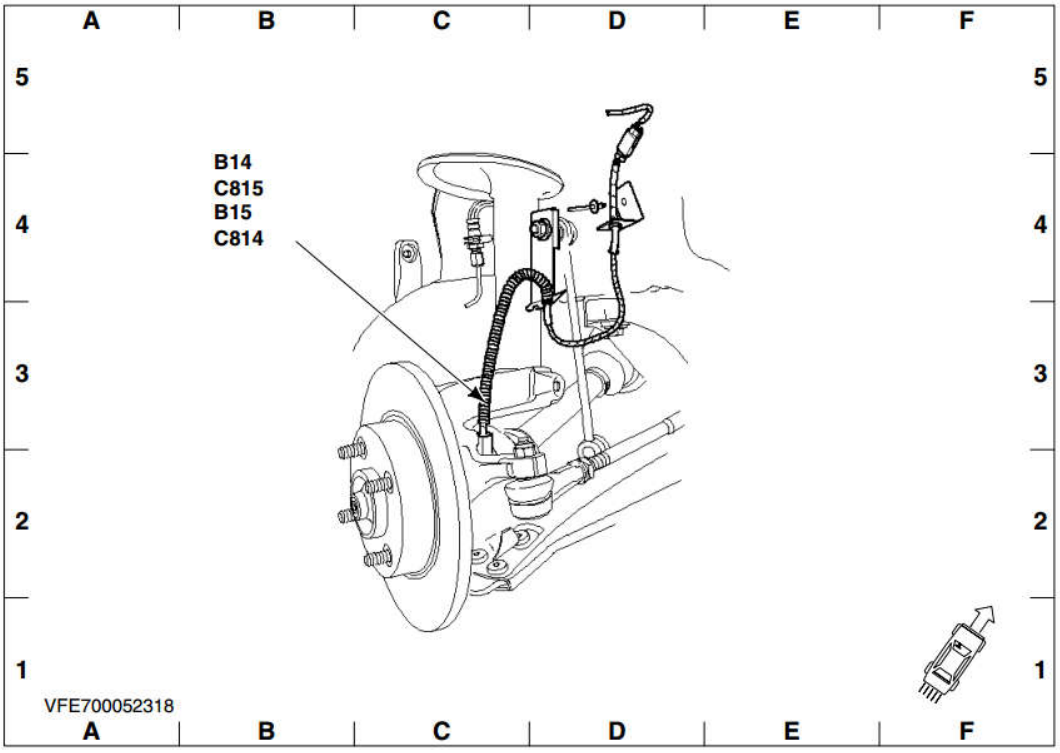
1.3 Conexión sensores ABS



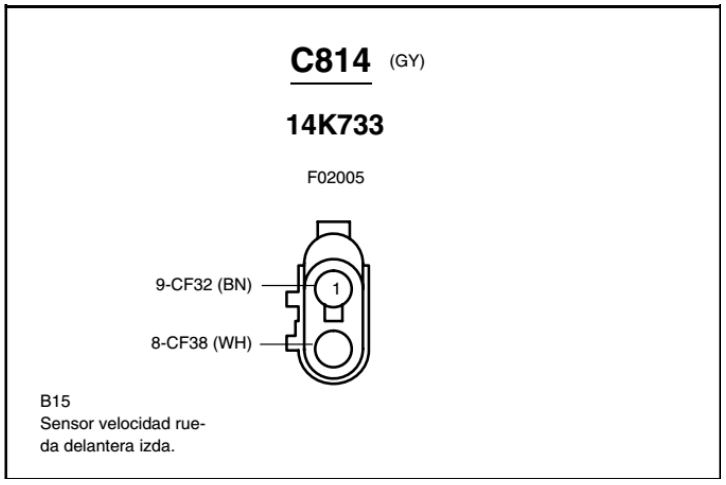
1.4 Situación del sensor izquierdo de ABS



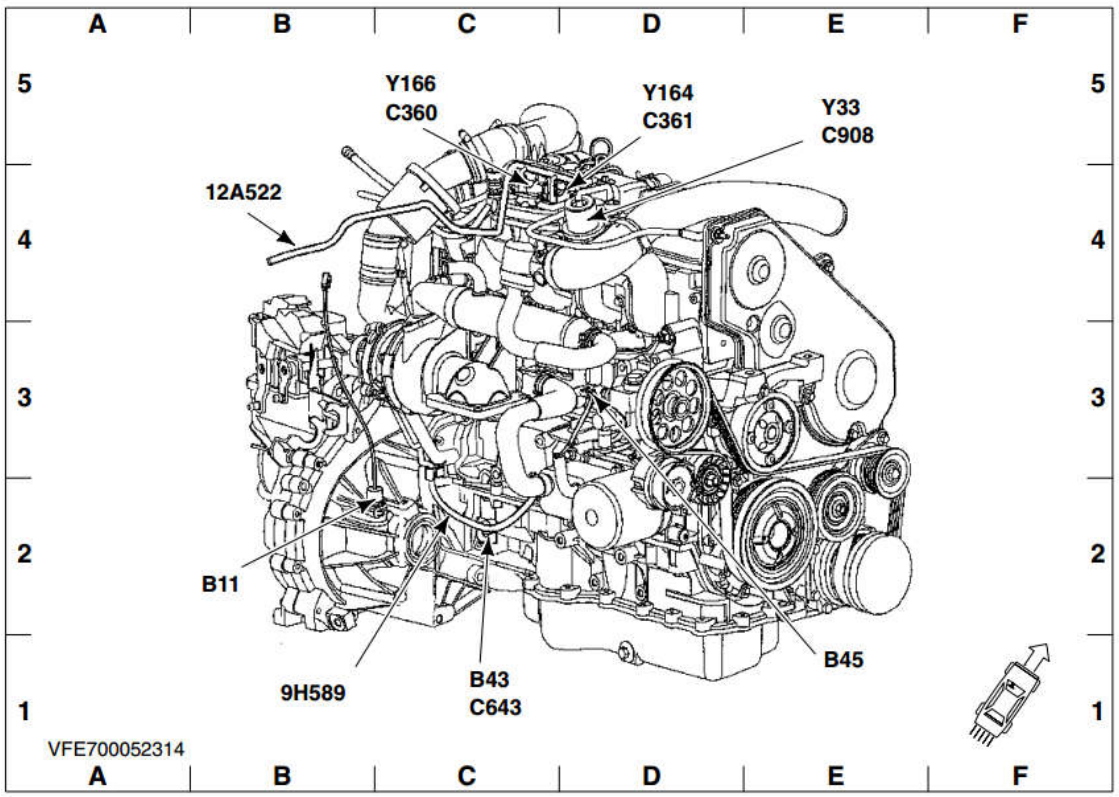
1.5 Situación del sensor izquierdo de ABS en mangueta de dirección



1.6 Conector del sensor izquierdo de ABS



1.7 Situación del sensor de velocidad VSS



1.8 Conexión del sensor de velocidad VSS al módulo de control del vehículo

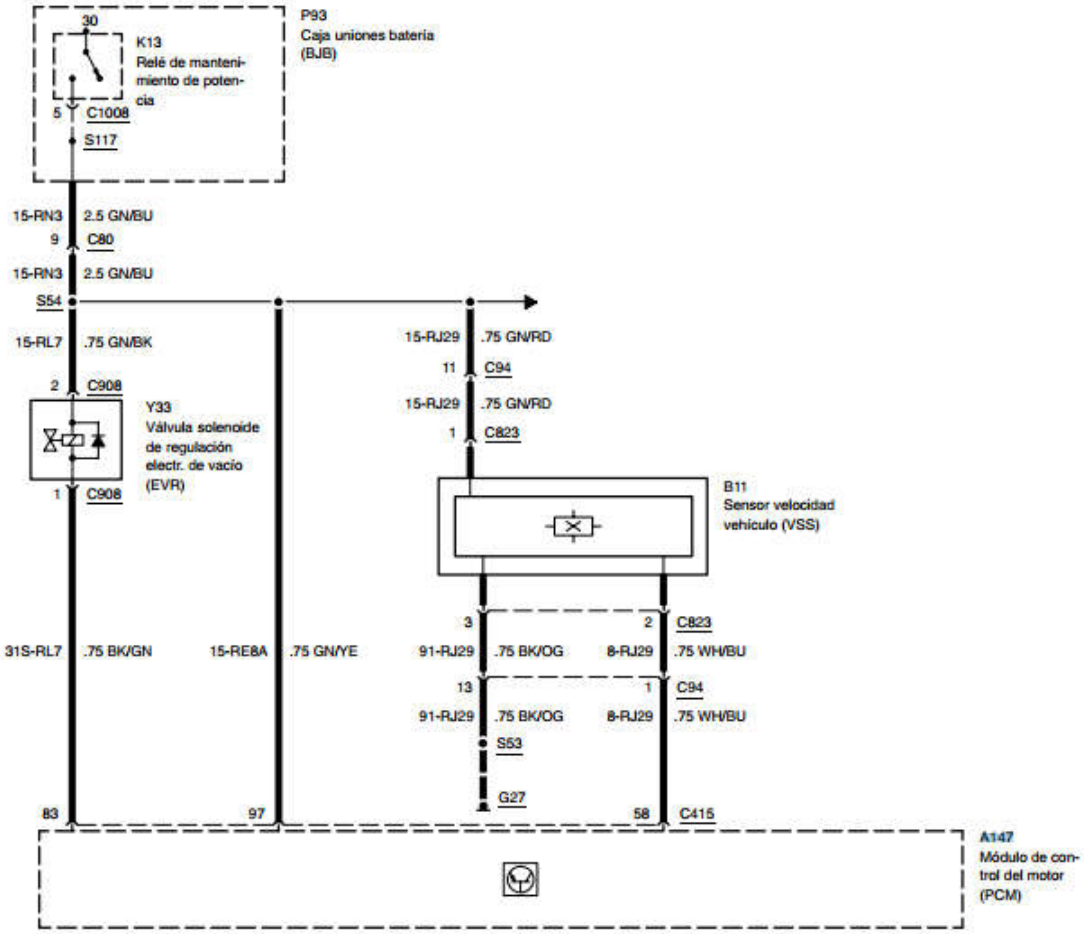
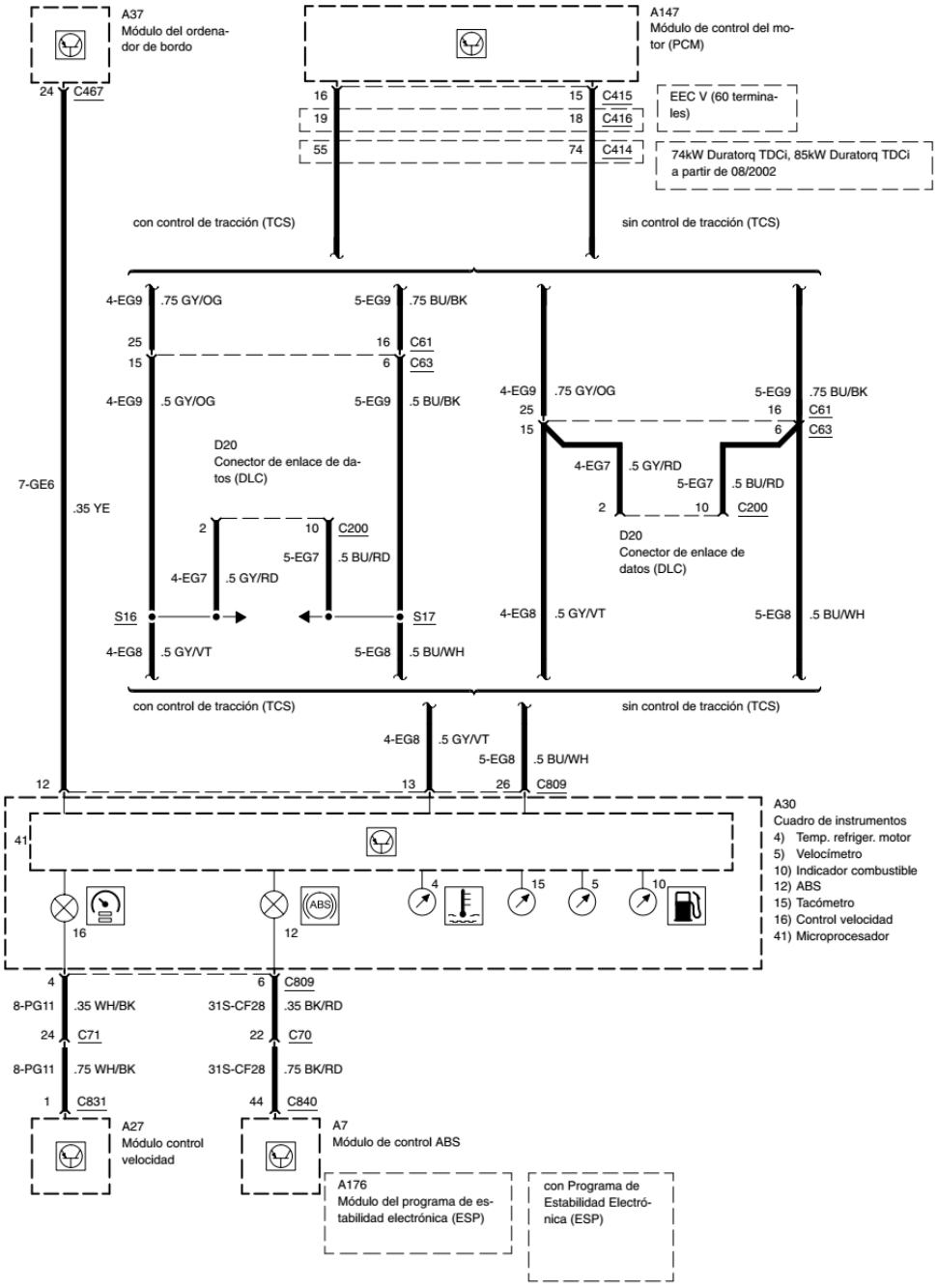


Ilustración 1: Conexiones del sensor VSS al módulo de control del vehículo

1.9 Esquema de tacómetro en el cuadro de instrumentos



UNIVERSIDAD DE LA RIOJA

ANEXO II: CÓDIGO DE ARDUINO

Desarrollo de Sistema de Control de Velocidad para automóvil sincronizado con la base de datos de cinemómetros de la DGT

UNIVERSIDAD DE LA RIOJA
GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Firma del presente documento:

Iván Peña Moreno

UNIVESIDAD DE LA RIOJA

Logroño, 3 de Marzo de 2019

ANEXO II: CÓDIGO DE ARDUINO

1. Código Arduino de la prueba acelerador.

```
// to the pins used:
const int analogInPin = A0;  // Entrada analógica para el
potenciómetro de consigna
const int analogOutPin3 = 3; // Salidas PWM para la clonación del
acelerador
const int analogOutPin5 = 5;
const int analogOutPin6 = 6;

int sensorValue = 0;        // Asigna valores iniciales
int outputValue3 = 0;
int outputValue5 = 0;
int outputValue6 = 0;

void setup() {

  Serial.begin(9600);       // inicializa el puerto serie a velocidad
de 9600 bps
}

void loop() {
  // lectura del valor analógico
  sensor Value = analogRead(analogInPin);
  // function map, escala los malores asignando minimos y maximos
  outputValue3 = map(sensorValue, 0, 1023, 66, 199);
//65.79=1.29*255/5  pin 4
  outputValue5 = map(sensorValue, 0, 1023,221,43 ); // pin 5
  outputValue6 = map(sensorValue, 0, 1023, 36, 171); // pin 7
  // escribe el valor analogico en la salida corespondiente como valor
PWM
  analogWrite(analogOutPin3, outputValue3);
  analogWrite(analogOutPin5, outputValue5);
  analogWrite(analogOutPin6, outputValue6);
  //Muestra en pantalla a través de Puerto serie
  Serial.print("sensor = ");
  Serial.print(sensorValue);
  Serial.print("\t");
  Serial.print(outputValue3);
  Serial.print("\t");
  Serial.print(outputValue5);
  Serial.print("\t");
  Serial.println(outputValue6);


  delay(200);
}
```

2. Codigo de Aruino para la lectura de datos en el CAN Bus

```
#include <mcp_can.h>
#include <mcp_can_dfs.h>
#include <SPI.h>
INT32U canId = 0x000;
unsigned char len = 0;
unsigned char buf[8];
char str[20];
const int SPI_CS_PIN = 9;
MCP_CAN CAN(SPI_CS_PIN);
void setup()
{
Serial.begin(115200);
START_INIT:
if(CAN_OK == CAN.begin(CAN_500KBPS))
    {
        Serial.println("CAN BUS Shield init ok!");
    }
    else
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println("Init CAN BUS Shield again");
        delay(100);
        goto START_INIT;
    }
}
void loop(){
    if(CAN_MSGAVAIL == CAN.checkReceive())
    {
        CAN.readMsgBuf(&len, buf);
        canId = CAN.getCanId();

          Serial.print(canId);Serial.print(",");
           for(int i = 0; i<len; i++){
              Serial.print(buf[i]);Serial.print(",");
           }
           Serial.println();
   }

}
```

3. Lectura velocímetro con Arduino UNO

```
const int analogInPin0 = A0;   // Configura entradas analógicas
const int analogInPin1 = A1;
const int analogInPin2 = A2;
const int analogInPin3 = A3;

int sensorValue0 = 0;        // Asigna valores iniciales
int sensorValue1 = 0;
int sensorValue2 = 0;
int sensorValue3 = 0;

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {
  // Lectura de las entradas analógicas correspondientes a las bobinas
  sensorValue0 = analogRead(analogInPin0);
  sensorValue1 = analogRead(analogInPin1);
  sensorValue2 = analogRead(analogInPin2);
  sensorValue3 = analogRead(analogInPin3);


  // Envia datos a traves de comunicación serie para la visualización
y adquisición de datos:
  Serial.print("sensor0 = ");
  Serial.print(sensorValue0);
  Serial.print("\t");
  Serial.print(sensorValue1);
  Serial.print("\t");
  Serial.print(sensorValue2);
  Serial.print("\t");
  Serial.println(sensorValue3);


  // Tiempo de espera para el ciclo de scan

  delay(200);
}
```

4. Lectura velocímetro por tramos con Arduino UNO

```
// to the pins used:
const int analogInPin0 = A0;  // Configura entradas analógicas
const int analogInPin1 = A1;
const int analogInPin2 = A2;
const int analogInPin3 = A3;

int Bobina11 = 0;        // Asigna valores iniciales
int Bobina12 = 0;
int Bobina21 = 0;
int Bobina22 = 0;
int vel = 0;

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {

    while (1) {


  // Lectura de las entradas analógicas correspondientes a las bobinas
  Bobina11 = analogRead(analogInPin0);
  Bobina12 = analogRead(analogInPin1);
  Bobina21 = analogRead(analogInPin2);
  Bobina22 = analogRead(analogInPin3);


 //Función para la selección de los tramos dependiendo de los valores
de entrada procedentes de las bobinas
   if ( Bobina21 > Bobina22) {
    vel = (-0.0539*Bobina21 + 65);
  } else if ( Bobina22 > Bobina21) {
       if ( Bobina22 > Bobina12) {
             if (Bobina11 > Bobina12) {
               vel = (0.0571*Bobina11 + 130);
             } else {
               vel = (-0.0543*Bobina12 + 142);
                              }
      } else {
        vel = ( 0.0557 * Bobina22 + 53);
      }

  } else {
    vel = 200;
    }


  // Envia datos a traves de comunicación serie para la visualización
y adquisición de datos:
  Serial.print("sensor0 = ");
  Serial.print(Bobina11);
  Serial.print("\t");
  Serial.print(Bobina12);
  Serial.print("\t");
  Serial.print(Bobina21);
```

```
  Serial.print("\t");
  Serial.print(Bobina22);
  Serial.print("\t");
  Serial.println(vel);


  delay(200);
  }
}
```

5. Prueba de GPS con el módulo NEO-6M

```
#include <SoftwareSerial.h> //Libreria para generar otros pines
adicionales como pines de comunicación

const int RX = 3;
const int TX = 4;

SoftwareSerial gps(RX, TX); // Asignación de pines para la
comunicación GPS

void setup()
{
   Serial.begin(115200); //Configura velocidad para el envío de datos
al ordenador
   gps.begin(9600); //Configura velocidad para la comunicación con el
módulo GPS
}

void loop()
{
  //Funcion para imprimir en pantalla todos los carecteres que se van
reciviendo
   if (gps.available())
   {
      char data;
      data = gps.read();
      Serial.print(data);
   }
}
```

6. Prueba GPS con la librería TinyGPS.h

```
#include <SoftwareSerial.h>
#include <TinyGPS.h>

TinyGPS gps;
SoftwareSerial softSerial(4, 3);

void setup()
{
   Serial.begin(115200);
   softSerial.begin(9600);
}

void loop()
{
   bool newData = false;
   unsigned long chars;
   unsigned short sentences, failed;

   // Intentar recibir secuencia durante un segundo
   for (unsigned long start = millis(); millis() - start < 1000;)
   {
      while (softSerial.available())
      {
         char c = softSerial.read();
         if (gps.encode(c)) // Nueva secuencia recibida
            newData = true;
      }
   }

   if (newData)
   {
      float flat, flon;
      unsigned long age;
      gps.f_get_position(&flat, &flon, &age);
      Serial.print("LAT=");
      Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat,
6);
      Serial.print(" LON=");
      Serial.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon,
6);
      Serial.print(" SAT=");
      Serial.print(gps.satellites() == TinyGPS::GPS_INVALID_SATELLITES
? 0 : gps.satellites());
      Serial.print(" PREC=");
      Serial.print(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 :
gps.hdop());
   }

   gps.stats(&chars, &sentences, &failed);
   Serial.print(" CHARS=");
   Serial.print(chars);
   Serial.print(" SENTENCES=");
   Serial.print(sentences);
   Serial.print(" CSUM ERR=");
   Serial.println(failed);
}
```

7. Prueba Bluetooth HC-06

```
int char_blue=0;
int analogo=0;


void setup(){
  pinMode(13,OUTPUT);
  Serial.begin(9600);
  delay(1000);

 }
 void loop(){
 if(Serial.available()>0){            // Si el puerto serie esta
habilitadp
        char_blue = Serial.read();      // Lee lo que llega por el
puerto Serie
 }

 if(char_blue== '1'){                    // on/off de pin 13
      digitalWrite(13,HIGH);
 }
 if(char_blue== '2'){                    // on/off de pin 13
      digitalWrite(13,LOW);
 }


  if(char_blue== 'R'){       // envia todos los datos cuando la
aplicación lo solicita con el carácter R
  analogo=analogRead(A0);
  Serial.print(analogo);
  Serial.print("|");
  Serial.print("234");
  Serial.print("|");
  Serial.print("987");
  Serial.print("|");
  Serial.print(analogo);
  Serial.print("|");
  Serial.print("234");
  Serial.print("|");
  Serial.print("987");
  Serial.print("|");
  }
 char_blue=0;
```

8. Prueba mando Radio-CD

```
const int analogInPin6 = A6;

int valor_mando = 0;
int cuentaVS =0;
int cuentaVB =0;
int cuentaSS =0;
int cuentaSB =0;
int cuentamode =0;
boolean P_VS =false;
boolean P_VB =false;
boolean P_SS =false;
boolean P_SB =false;
boolean P_mode =false;
int errormando = 1;

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(115200);
}

void loop() {

    while (1) {

 valor_mando = analogRead(analogInPin6);

if (valor_mando < 100) { // Analiza boton pulsado y antirrebote
errormando = 1;} //Error por mando desconectado
else if (valor_mando < 300) { //Mando conectado
errormando = 0;
cuentaVS = 0;
cuentaVB = 0;
cuentaSS = 0;
cuentaSB = 0;
cuentamode = 0;
P_mode = false;
P_VS = false;
P_VB = false;
P_SS = false;
P_SB = false;}
else if (valor_mando < 570) { // Pulsador mode
  cuentamode = cuentamode+1;}
else if (valor_mando < 700) { //Pulsador Seek subir
  cuentaSS = cuentaSS+1;}
else if (valor_mando < 820) { //Pulsador Seek bajar
  cuentaSB = cuentaSB+1;}
else if (valor_mando < 930) { //Pulsador Velocidad subir
  cuentaVS = cuentaVS+1;}
else if (valor_mando < 1020) { //Pulsador Velocidad bajar
  cuentaVB = cuentaVB+1;}

if (cuentamode > 100)//Valida pulsacion
P_mode = true;
if (cuentaVS > 100)//Valida pulsacion
P_VS = true;
if (cuentaVB > 100)//Valida pulsacion
P_VB = true;
if (cuentaSS > 100)//Valida pulsacion
P_SS = true;
```

```
if (cuentaSB > 100)//Valida pulsacion
P_SB = true;

Serial.print(P_mode);
Serial.print(P_VS);
Serial.print(P_VB);
Serial.print(P_SS);
Serial.print(P_SB);
Serial.println(valor_mando);

  delay(100);
  }
}
```

9. Adquisición de datos en lazo abierto

```
// to the pins used:
const int analogInPin0 = A0;  // Configura entradas analógicas
const int analogInPin1 = A1;
const int analogInPin2 = A2;
const int analogInPin3 = A3;
const int analogInPin4 = A4;

int Bobina11 = 0;        // Asigna valores iniciales
int Bobina12 = 0;
int Bobina21 = 0;
int Bobina22 = 0;
int vel = 0;
int potacel = 0;
int acel = 0;

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {

    while (1) {


  // Lectura de las entradas analógicas correspondientes a las bobinas
  Bobina11 = analogRead(analogInPin0);
  Bobina12 = analogRead(analogInPin1);
  Bobina21 = analogRead(analogInPin2);
  Bobina22 = analogRead(analogInPin3);
  // Lectura de las entradas analógica del pin4 del aclerador
  potacel = analogRead(analogInPin4);

  acel = map(potacel, 220, 750, 0, 100); //Escalado del pin 4 de la
acción del acelerador, para obtener calores de 0 a 100%.


 //Función para la selección de los tramos dependiendo de los valores
de entrada procedentes de las bobinas
   if ( Bobina21 > Bobina22) {
    vel = (-0.0539*Bobina21 + 65);
  } else if ( Bobina22 > Bobina21) {
       if ( Bobina22 > Bobina12) {
             if (Bobina11 > Bobina12) {
               vel = (0.0571*Bobina11 + 130);
             } else {
               vel = (-0.0543*Bobina12 + 142);
                           }
      } else {
        vel = ( 0.0557 * Bobina22 + 53);
      }

  } else {
    vel = 200;
    }
```

```
  // Envia datos a traves de comunicación serie para la visualización
y adquisición de datos:
  Serial.print("sensor0 = ");
  Serial.print(Bobina11);
  Serial.print("\t");
  Serial.print(Bobina12);
  Serial.print("\t");
  Serial.print(Bobina21);
  Serial.print("\t");
  Serial.print(Bobina22);
  Serial.print("\t");
  Serial.print(acel);
  Serial.print("\t");
  Serial.println(vel);

//Tiempo de espera para ralizar lectura de datos cada 50ms
aproximadamente
 delay(50);
  }
}
```

10. Programa final de Arduino Mega

```
const int analogInPin0 = A0;  // Analog input pin that the
potentiometer is attached to
const int analogInPin1 = A1;
const int analogInPin2 = A2;
const int analogInPin3 = A3;

int Bobina11 = 0;        // value read from the pot
int Bobina12 = 0;
int Bobina21 = 0;
int Bobina22 = 0;
int vel = 0;

const int acel_in_4 = A4; //Pin 4 del acelerador, lectura acelerador
const int acel_in_5 = A5; //Pin 5 del acelerador, lectura acelerador
const int acel_in_7 = A7; //Pin 7 del acelerador, lectura acelerador
const int analogOutPin2 = 2; // Analog output pin that the LED is
attached to
const int analogOutPin3 = 3;
const int analogOutPin4 = 4;

int acelerador = 0;
int acelerador_in = 0;
int acelerador_in_5 = 0;
int acelerador_in_7 = 0;
int acelerador_out = 0;// value read from the pot
int outputValue2 = 0;        // value output to the PWM (analog out)
int outputValue3 = 0;
int outputValue4 = 0;
bool error_acelerador = false;

const int analogInPin6 = A6;

int valor_mando = 0;
int cuentaVS =0;
int cuentaVB =0;
```

```
int cuentaSS =0;
int cuentaSB =0;
int cuentamode =0;
boolean P_VS =false;
boolean P_VB =false;
boolean P_SS =false;
boolean P_SB =false;
boolean P_mode_L =false;
boolean P_mode_C =false;
int errormando = 1;

int char_blue=0;

int Input, Output, Setpoint; //No hay valores negarivos, y estos
valores negativos perjudicarian
int Outputint =0;
float ITerm, lastInput;
//double kp, ki, kd;
int t_muestreo_control = 200;
float kp=2;
float ki=0.1;
float error =0;
int t_control = 0;
unsigned long f_control;

int t_muestreo_gps = 500;
#include <TinyGPS.h>
TinyGPS gps;
bool newData = false;
unsigned long chars;
unsigned short sentences, failed;
float flat, flon = 0;
unsigned long age;
float velgps = 0;
float direccion = 0;
float altitud = 0;
int t_gps = 0;
unsigned long tf_gps = 0;
float d_x = 0;
float d_y = 0;
float R = 6371000;
float distancia_radar =0;

bool freno = true;
bool embrague = true;
int estado =0;
int t_scan = 0;
unsigned long ultimo_scan = 0;
bool actcontrol=false;
int modo = 1;
int K_frenaradar = 3; //Factor para el calculo de distancia de frenada
int frenaradar = 3; //Distancia de frenada al radar
int lim_vel = 50; //Limite de velocidad en la via (lista)
int set_lim_vel = 0; //Consigna de limite de velicidad
int ult_distancia_radar = 0;  //Ulmima distancia a radar fijo, para
detectar el alejamiento del mismo



void setup() {
  // initialize serial communications at 9600 bps:
```

```
  Serial.begin(115200);
  Serial2.begin(9600); //GPS
  Serial3.begin(9600);   //Bluetooth
  pinMode(13,OUTPUT);
  delay(100);
}

void mando() {

 valor_mando = analogRead(analogInPin6);

if (valor_mando < 100) { // Analiza boton pulsado y antirrebote
errormando = 1;} //Error por mando desconectado
else if (valor_mando < 300) { //Mando conectado
errormando = 0;
cuentaVS = 0;
cuentaVB = 0;
cuentaSS = 0;
cuentaSB = 0;
cuentamode = 0;
P_mode_L = false;
P_mode_C = false;
P_VS = false;
P_VB = false;
P_SS = false;
P_SB = false;}
else if (valor_mando < 570) { // Pulsador mode
  cuentamode = cuentamode+1;}
else if (valor_mando < 700) { //Pulsador Seek subir
  cuentaSS = cuentaSS+1;
  P_SS = false;}
else if (valor_mando < 820) { //Pulsador Seek bajar
  cuentaSB = cuentaSB+1;
  P_SB = false;}
else if (valor_mando < 930) { //Pulsador Velocidad subir
  cuentaVS = cuentaVS+1;
  P_VS = false;}              //Condicion para generar pulsos con
entrada activa
else if (valor_mando < 1020) { //Pulsador Velocidad bajar
  cuentaVB = cuentaVB+1;      //Condicion para generar pulsos con
entrada activa
  P_VB = false;}

if (cuentamode > 200){//Valida pulsacion
P_mode_C = false;
P_mode_L = true;}
if (cuentamode == 10)//Valida pulsacion
P_mode_C = true;
if (cuentaVS > 60){//Valida pulsacion
P_VS = true;
cuentaVS = 0;}
if (cuentaVB > 60){//Valida pulsacion
P_VB = true;
cuentaVB = 0;}
if (cuentaSS == 200)//Valida pulsacion en único ciclo de programa
P_SS = true;
if (cuentaSB == 200)//Valida pulsacion en único ciclo de programa
P_SB = true;
}

void acelera() {
```

```
// read the analog in value:
acelerador_in = map(analogRead(acel_in_4), 220, 750, 0, 100); //
Conexión a pin 4 del acelerador, escalado %
acelerador_in_5 = map(analogRead(acel_in_5), 220, 750, 0, 100); //
Conexión a pin 5 del acelerador, escalado %
acelerador_in_7 = map(analogRead(acel_in_7), 220, 750, 0, 100); //
Conexión a pin 7 del acelerador, escalado %

// Función para la seguridad del correcto funcionamiento, comprueba
que no exista una desviación de 10% entre señales del acelerador.
if ((((acelerador_in + acelerador_in_5 + acelerador_in_7)/3)-
acelerador_in) > 10)
error_acelerador = true;
if ((((acelerador_in + acelerador_in_5 + acelerador_in_7)/3)-
acelerador_in) > 10)
error_acelerador = true;
  // map it to the range of the analog out:
  outputValue2 = map(acelerador_out, 0, 100, 66, 199);
//65.79=1.29*255/5  pin 4
  outputValue3 = map(acelerador_out, 0, 100,221,43 ); // pin 5
  outputValue4 = map(acelerador_out, 0, 100, 36, 171); // pin 7
  // change the analog out value:
  analogWrite(analogOutPin2, outputValue2);
  analogWrite(analogOutPin3, outputValue3);
  analogWrite(analogOutPin4, outputValue4);
}

void velocimetro() {

   // read the analog in value:
  Bobina11 = analogRead(analogInPin0);
  Bobina12 = analogRead(analogInPin1);
  Bobina21 = analogRead(analogInPin2);
  Bobina22 = analogRead(analogInPin3);

   if ( Bobina21 > Bobina22) {
    vel = (-0.0539*Bobina21 + 65);}
      else if ( Bobina22 > Bobina21) {
        if ( Bobina22 > Bobina12) {
              if (Bobina11 > Bobina12) {
                vel = (0.0571*Bobina11 + 130);
              } else {
                vel = (-0.0543*Bobina12 + 142);}
         }
        else {
        vel = ( 0.0557 * Bobina22 + 53);}

  } else {
    vel = 2000;
    }
}


 void bluetooht(){
 if(Serial3.available()>0){           // Si el puerto serie esta
habilitado
        char_blue = Serial3.read();      // Lee lo que llega por el
puerto Serie
 }

 if(char_blue== '1'){                    // on/off de pin 136
```

```
      digitalWrite(13,HIGH);
 }
 if(char_blue== '2'){                    // on/off de pin 13
      digitalWrite(13,LOW);
 }

  if(char_blue== 'R'){       // envia todos los datos cuando la
aplicación lo solicita (R)
  Serial3.print(estado);
  Serial3.print("|");
  Serial3.print(acelerador_out);
  Serial3.print("|");
  Serial3.print(vel);
  Serial3.print("|");
  Serial3.print(distancia_radar);
  Serial3.print("|");
  Serial3.print(error,2);
  Serial3.print("|");
  Serial3.print(ITerm,2);
  Serial3.print("|");
   }
 char_blue=0;
 }


 void lazo_control()
{
//Función para la ejecución por periodos de muestreo del control
t_control = (ultimo_scan - f_control);
if(t_control > t_muestreo_control)
{
// Calcula el error del lazo de control.
error = Setpoint - vel;
ITerm = ITerm + (ki * error);
if (ITerm > 20){
  ITerm = 20; }

// Calculamos la función de salida del PI.
Output = kp * error + ITerm;
Outputint = Output; // convierte el decimal a entero
actcontrol = true; //Refistro de activación dela función control. Bit
para las acciones temporizadas (serial PC)

lastInput = Input;
f_control = ultimo_scan;

if (Outputint > 75){ //Aunti windup
Outputint = 75;}
if (Outputint < 4){ //Aunti windup
Outputint = 4;}
// Guardamos el valor de algunas variables para el próximo recálculo.

}
}

float dif_LAT_ultima = 0;
float dif_LON_ultima = 0;
int p = 0;

// Radar {Lobete, Sequero, Sequero2, Duques, Universidad}
```

```
float ref_LAT[] = { 42.465729, 42.458930, 42.458887, 42.434203,
42.429933};
float ref_LON[] ={ -2.422874, -2.432625, -2.452598, -2.2782757, -
2.277204};
float ref_direccion[]={0, 0, 0, 0, 0}; //Orientación en la que detecta
el radar, 0 para ambos sentidos

void lista_gps()
{
dif_LAT_ultima = flat - ref_LAT[p];//Guarda la diferencia con puntero
ultima
dif_LON_ultima = flon - ref_LON[p];//Guarda la diferencia con puntero
ultima

if((flat - ref_LAT[p++]) < dif_LAT_ultima){
p = p++; //Incrementar el puntero para la coordeana mas proxima
}
if((flon - ref_LAT[p--]) < dif_LAT_ultima){
p = p--; //Decrementar el puntero para la coordeana mas proxima
}

dif_LAT_ultima = flat - ref_LAT[p];//Guarda la diferencia con puntero
actual
dif_LON_ultima = flon - ref_LON[p];//Guarda la diferencia con puntero
actual

}

void f_gps()
{


   // Recibir secuencia cada segundo
t_gps = (ultimo_scan - tf_gps);
if(t_gps>=t_muestreo_gps)
{
      while (Serial2.available())
      {
         char c = Serial2.read();
         if (gps.encode(c)){ // Nueva secuencia recibida
            newData = true;
            tf_gps = millis();}
      }
   }
   if (newData){
    flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6;
    flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6;
    //Actualización del punto más proximo de gps de la lista
    lista_gps();
    //Calculo de distancia al punto de referencia
    d_x = R * sin((dif_LAT_ultima)* PI/180);
    d_y = R * sin((dif_LON_ultima)* PI/180);
    distancia_radar = sqrt(d_y*d_y+d_x*d_x);
//

//
    }

     gps.stats(&chars, &sentences, &failed);
//   Serial.print(" CHARS=");
//   Serial.print(chars);
```

```
//   Serial.print(" SENTENCES="); // mayor que cero empieza a emviar,
sino incementa no ha actualizado
//   Serial.print(sentences);
//   Serial.print(" CSUM ERR=");
//   Serial.println(failed);

   if (newData)
   {

      gps.f_get_position(&flat, &flon, &age);
      velgps = gps.f_speed_kmph();
      direccion = gps.f_course();
      altitud = gps.f_altitude();

//      Serial.print("LAT=");
//      Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 :
flat, 6);
//      Serial.print(" LON=");
//      Serial.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 :
flon, 6);
//      Serial.print(" SAT=");
//      Serial.print(gps.satellites() ==
TinyGPS::GPS_INVALID_SATELLITES ? 0 : gps.satellites());
//      Serial.print(" PREC=");
//      Serial.print(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 :
gps.hdop());
//      Serial.print(" Km/h=");
//      Serial.print(velgps);
//      Serial.print(" Direcciion=");
//      Serial.print(direccion);
//      Serial.print(" Altitud=");
//      Serial.print(" Distancia x =");
//      Serial.print(d_x);
//      Serial.print(" Distancia y =");
//      Serial.print(d_y);
//      Serial.print(" Distancia radar =");
//      Serial.print(distancia_radar);

   }


}

void loop(){
  f_gps();
  lazo_control();
  bluetooht();
  acelera();
  velocimetro();
  mando();



  if (freno == true){
  acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
  estado = 0;}

  if (embrague == true){
  acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
```

```
  estado = 0;}

  if (P_mode_C == true){  //Desconexión de sistemas automaticos
  acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
  estado = 0;}


  switch (estado) {   //Maquina de estados para los modos
funcionamiento

    case 0:
    acelerador_out = acelerador_in;
    Setpoint = vel;

    if (P_mode_L == true){
    estado = modo;

    }
    else{
    estado = 0;
    }
    break;




    case 1://Estado activo.Control por proximidad a radar por GPS
    modo = 1;

    frenaradar = vel * K_frenaradar;

    if (distancia_radar > ult_distancia_radar){ //Función en caso de
alejarse del punto de radar fijo, dejar control automatico.
    ult_distancia_radar = distancia_radar; //Se actualiza ultima señal
de radar
    acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
    Setpoint = vel; //El controlador sigue funcionado para evitar
perturbaciones, con consigna igual a la velocidad real.
    }

    else if ( (distancia_radar < frenaradar) and (vel > lim_vel) ){
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = lim_vel; //Consigna de limite de velocidad referida
a la velocidad de la via
        ult_distancia_radar = distancia_radar; //Se actualiza ultima
señal de radar
        }
      else { //Desconexión de sistemas automaticos
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        Setpoint = lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
        ult_distancia_radar = distancia_radar; //Se actualiza ultima
señal de radar
        }
    }
```

```
    else {
    ult_distancia_radar = distancia_radar;  //Se actualiza ultima
señal de radar
    acelerador_out = acelerador_in;  //Señal de clonación igual a
señal real de acelerador.
    Setpoint = vel; //El controlador sigue funcionado para evitar
perturbaciones, con consigna igual a la velocidad real.
    }

    if (P_SS == true){  //Cambio de modo
    modo =2;
    set_lim_vel = vel;
    estado = 2;}

    if (P_SB == true){  //Cambio de modo
    modo =5;
    estado = 5;}

    if (P_mode_C == true){  //Desconexión de sistemas automaticos
    acelerador_out = acelerador_in;
    estado = 0;}

    break;


    case 2://Estado activo. Limite de velocidad + control por
proximidad a radar por GPS
    modo = 2;

    frenaradar = vel * K_frenaradar;


    if ((distancia_radar > ult_distancia_radar) and (vel <
set_lim_vel)){ //Función en caso de alejarse del punto de radar fijo y
que la velocidad sea menor a la consigna limite de velocidad,
desactivar control
    ult_distancia_radar = distancia_radar; //Se actualiza ultima señal
de radar
    acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
    Setpoint = set_lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
    }

    else if ((vel >= lim_vel) and (distancia_radar < frenaradar)){
//Función para para el paso por radar fijo por GPS
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = lim_vel; //Consigna de limite de velocidad referida
al limite seleccionado
        ult_distancia_radar = distancia_radar;
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        Setpoint = lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
        ult_distancia_radar = distancia_radar;
        }
    }
```

```
    else if ( vel > set_lim_vel ){ //Función para le control con
limitador de velocidad.
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = set_lim_vel; //Consigna de limite de velocidad
referida a la velocidad de la via
        ult_distancia_radar = distancia_radar;
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        Setpoint = set_lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
        ult_distancia_radar = distancia_radar;
        }
    }

    else {
    ult_distancia_radar = distancia_radar;
    acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
    Setpoint = set_lim_vel;  //En modo reposo consigna límite, así el
control esta estable para le uso
    }

    if (P_SS == true){  //Cambio de modo
    modo =3;
    Setpoint = vel;
    estado = 3;}

    if (P_SB == true){  //Cambio de modo
    modo =1;
    estado = 1;}

    if (P_VS == true){  //Guarda consigna, incrementa o decrementa
      set_lim_vel = set_lim_vel + 1;}
    if (P_VB == true){
      set_lim_vel = set_lim_vel - 1;}

    if (P_mode_C == true){  //Desconexión de sistemas automaticos
    acelerador_out = acelerador_in;
    estado = 0;}

    break;


    case 3://Estado activo. Control de crucero + control por
proximidad a radar por GPS
    modo = 3;

    frenaradar = vel * K_frenaradar;

    if (distancia_radar > ult_distancia_radar){ //Función en caso de
alejarse del punto de radar fijo, dejar control por proximidad
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        ult_distancia_radar = distancia_radar;
        }
```

```
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        ult_distancia_radar = distancia_radar;
        }
    }

    else if ( (distancia_radar < frenaradar) and (vel > lim_vel) ){
//Función para le control de velocidad por proximidad a GPS
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = lim_vel; //Consigna de limite de velocidad referida
a la velocidad de la via
        ult_distancia_radar = distancia_radar;
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        Setpoint = lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
        ult_distancia_radar = distancia_radar;
        }
    }

    else {    //Sigue el control de cruzero con la consigna del radar
pasado, se subirá de forma manual la velocidad para ello.
    ult_distancia_radar = distancia_radar;
    acelerador_out = Outputint; //Señal de clonación controlada por el
lazo de control
      if (P_VS == true){  //Guarda consigna de control de velocidad,
incrementa o decrementa
      Setpoint = Setpoint + 1;}
      if (P_VB == true){
      Setpoint = Setpoint - 1;}
    }

    if (P_SS == true){  //Cambio de modo
    modo =4;
    Setpoint = vel;
    estado = 4;}

    if (P_SB == true){  //Cambio de modo
    modo =2;
    set_lim_vel = vel;
    estado = 2;}


    if (P_mode_C == true){  //Desconexión de sistemas automaticos
    acelerador_out = acelerador_in;
    estado = 0;}

    break;

    case 4: //Estado activo. Control de crucero

    modo = 4;
    acelerador_out = Outputint;   //Señal de clonación controlada por
el lazo de control

    if (P_SS == true){  //Cambio de modo
```

```
    modo =5;
    set_lim_vel = vel;
    estado = 5;}

    if (P_SB == true){  //Cambio de modo
    modo =3;
    Setpoint = vel;
    estado = 3;}

    if (P_VS == true){  //Guarda consigna, incrementa o decrementa
      Setpoint = Setpoint + 1;}
    if (P_VB == true){
      Setpoint = Setpoint - 1;}

    if (P_mode_C == true){  //Desconexión de sistemas automaticos
    acelerador_out = acelerador_in;
    estado = 0;}

    break;

    case 5: //Estado activo. Control de limitador de velocidad

    modo =5;

    if (vel >= set_lim_vel){ //Función para el control de limitador de
velocidad
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = set_lim_vel; //Consigna de limite de velocidad
referida al limite seleccionado
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        Setpoint = set_lim_vel;//Consigna de limite de velocidad
referida al limite seleccionado
        }
    }

    else {
    acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
    Setpoint = set_lim_vel;  //En modo reposo consigna límite, así el
control esta estable para le uso
    }

    if (P_SS == true){  //Cambio de modo
    modo =1;
    estado = 1;}

    if (P_SB == true){  //Cambio de modo
    modo =4;
    Setpoint = vel;
    estado = 4;}

    if (P_VS == true){  //Guarda consigna, incrementa o decrementa
      set_lim_vel = set_lim_vel + 1;}
    if (P_VB == true){
      set_lim_vel = set_lim_vel - 1;}
```

```
    if (P_mode_C == true){  //Desconexión de sistemas automaticos
    acelerador_out = acelerador_in;
    estado = 0;}

    break;

      }

  if (P_mode_C == true){
  acelerador_out = acelerador_in;
  estado = 0;}

  t_scan = millis() - ultimo_scan;
  ultimo_scan = millis();


  if (actcontrol == true){    //Parámetros de control a través de
monitot serial
//  Serial.print("Activa consigna: ");
//  Serial.print(estado);
//  Serial.print("\tConsigna: ");
//  Serial.print(Setpoint);
//  Serial.print("\tAcelerador: ");
//  Serial.print(acelerador_out);
//  Serial.print("\tvelocidad: ");
//  Serial.print(vel);
//  Serial.print("\t scan: ");
//  Serial.println(t_scan);
//  Serial.print("\t control: ");
//  Serial.println(t_control);
//  Serial.print("\t error: ");
//  Serial.println(error,3);
//  Serial.print("\t Iterm: ");
//  Serial.println(ITerm,3);
//  Serial.print("\t output: ");
//  Serial.println(Output);
  actcontrol = false;
  }
  }
```

10.1 Máquina de estados

```
 if (freno == true){
  acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
  estado = 0;}

  if (embrague == true){
  acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
  estado = 0;}

  if (P_mode_C == true){  //Desconexión de sistemas automaticos
  acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
  estado = 0;}


  switch (estado) {   //Maquina de estados para los modos
funcionamiento

    case 0:
    acelerador_out = acelerador_in;
    Setpoint = vel;

    if (P_mode_L == true){
    estado = modo;

    }
    else{
    estado = 0;
    }
    break;




    case 1://Estado activo.Control por proximidad a radar por GPS
    modo = 1;

    frenaradar = vel * K_frenaradar;

    if (distancia_radar > ult_distancia_radar){ //Función en caso de
alejarse del punto de radar fijo, dejar control automatico.
    ult_distancia_radar = distancia_radar; //Se actualiza ultima señal
de radar
    acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
    Setpoint = vel; //El controlador sigue funcionado para evitar
perturbaciones, con consigna igual a la velocidad real.
    }

    else if ( (distancia_radar < frenaradar) and (vel > lim_vel) ){
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = lim_vel; //Consigna de limite de velocidad referida
a la velocidad de la via
```

```
        ult_distancia_radar = distancia_radar; //Se actualiza ultima
señal de radar
        }
      else { //Desconexión de sistemas automaticos
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        Setpoint = lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
        ult_distancia_radar = distancia_radar; //Se actualiza ultima
señal de radar
        }
    }

    else {
    ult_distancia_radar = distancia_radar;  //Se actualiza ultima
señal de radar
    acelerador_out = acelerador_in;  //Señal de clonación igual a
señal real de acelerador.
    Setpoint = vel; //El controlador sigue funcionado para evitar
perturbaciones, con consigna igual a la velocidad real.
    }

    if (P_SS == true){  //Cambio de modo
    modo =2;
    set_lim_vel = vel;
    estado = 2;}

    if (P_SB == true){  //Cambio de modo
    modo =5;
    estado = 5;}

    if (P_mode_C == true){  //Desconexión de sistemas automaticos
    acelerador_out = acelerador_in;
    estado = 0;}

    break;


    case 2://Estado activo. Limite de velocidad + control por
proximidad a radar por GPS
    modo = 2;

    frenaradar = vel * K_frenaradar;


    if ((distancia_radar > ult_distancia_radar) and (vel <
set_lim_vel)){ //Función en caso de alejarse del punto de radar fijo y
que la velocidad sea menor a la consigna limite de velocidad,
desactivar control
    ult_distancia_radar = distancia_radar; //Se actualiza ultima señal
de radar
    acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
    Setpoint = set_lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
    }

    else if ((vel >= lim_vel) and (distancia_radar < frenaradar)){
//Función para para el paso por radar fijo por GPS
      if (acelerador_in > Outputint ){
```

```
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = lim_vel; //Consigna de limite de velocidad referida
al limite seleccionado
        ult_distancia_radar = distancia_radar;
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        Setpoint = lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
        ult_distancia_radar = distancia_radar;
        }
    }

    else if ( vel > set_lim_vel ){ //Función para le control con
limitador de velocidad.
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = set_lim_vel; //Consigna de limite de velocidad
referida a la velocidad de la via
        ult_distancia_radar = distancia_radar;
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        Setpoint = set_lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
        ult_distancia_radar = distancia_radar;
        }
    }

    else {
    ult_distancia_radar = distancia_radar;
    acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
    Setpoint = set_lim_vel;  //En modo reposo consigna límite, así el
control esta estable para le uso
    }

    if (P_SS == true){  //Cambio de modo
    modo =3;
    Setpoint = vel;
    estado = 3;}

    if (P_SB == true){  //Cambio de modo
    modo =1;
    estado = 1;}

    if (P_VS == true){  //Guarda consigna, incrementa o decrementa
      set_lim_vel = set_lim_vel + 1;}
    if (P_VB == true){
      set_lim_vel = set_lim_vel - 1;}

    if (P_mode_C == true){  //Desconexión de sistemas automaticos
    acelerador_out = acelerador_in;
    estado = 0;}

    break;
```

```
    case 3://Estado activo. Control de crucero + control por
proximidad a radar por GPS
    modo = 3;

    frenaradar = vel * K_frenaradar;

    if (distancia_radar > ult_distancia_radar){ //Función en caso de
alejarse del punto de radar fijo, dejar control por proximidad
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        ult_distancia_radar = distancia_radar;
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        ult_distancia_radar = distancia_radar;
        }
    }

    else if ( (distancia_radar < frenaradar) and (vel > lim_vel) ){
//Función para le control de velocidad por proximidad a GPS
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = lim_vel; //Consigna de limite de velocidad referida
a la velocidad de la vía
        ult_distancia_radar = distancia_radar;
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        Setpoint = lim_vel; //El controlador sigue funcionado para
evitar perturbaciones
        ult_distancia_radar = distancia_radar;
        }
    }

    else {    //Sigue el control de cruzero con la consigna del radar
pasado, se subirá de forma manual la velocidad para ello.
    ult_distancia_radar = distancia_radar;
    acelerador_out = Outputint; //Señal de clonación controlada por el
lazo de control
      if (P_VS == true){  //Guarda consigna de control de velocidad,
incrementa o decrementa
      Setpoint = Setpoint + 1;}
      if (P_VB == true){
      Setpoint = Setpoint - 1;}
    }

    if (P_SS == true){  //Cambio de modo
    modo =4;
    Setpoint = vel;
    estado = 4;}

    if (P_SB == true){  //Cambio de modo
    modo =2;
    set_lim_vel = vel;
    estado = 2;}
```

```
    if (P_mode_C == true){  //Desconexión de sistemas automaticos
    acelerador_out = acelerador_in;
    estado = 0;}

    break;

    case 4: //Estado activo. Control de crucero

    modo = 4;
    acelerador_out = Outputint;   //Señal de clonación controlada por
el lazo de control

    if (P_SS == true){  //Cambio de modo
    modo =5;
    set_lim_vel = vel;
    estado = 5;}

    if (P_SB == true){  //Cambio de modo
    modo =3;
    Setpoint = vel;
    estado = 3;}

    if (P_VS == true){  //Guarda consigna, incrementa o decrementa
      Setpoint = Setpoint + 1;}
    if (P_VB == true){
      Setpoint = Setpoint - 1;}

    if (P_mode_C == true){  //Desconexión de sistemas automaticos
    acelerador_out = acelerador_in;
    estado = 0;}

    break;

    case 5: //Estado activo. Control de limitador de velocidad

    modo =5;

    if (vel >= set_lim_vel){ //Función para el control de limitador de
velocidad
      if (acelerador_in > Outputint ){
        acelerador_out = Outputint; //Señal de clonación controlada
por el lazo de control
        Setpoint = set_lim_vel; //Consigna de limite de velocidad
referida al limite seleccionado
        }
      else {
        acelerador_out = acelerador_in; //Señal de clonación igual a
señal real de acelerador.
        Setpoint = set_lim_vel;//Consigna de limite de velocidad
referida al limite seleccionado
        }
    }

    else {
    acelerador_out = acelerador_in; //Señal de clonación igual a señal
real de acelerador.
    Setpoint = set_lim_vel;  //En modo reposo consigna límite, así el
control esta estable para le uso
    }
```

```
if (P_SS == true){  //Cambio de modo
modo =1;
estado = 1;}

if (P_SB == true){  //Cambio de modo
modo =4;
Setpoint = vel;
estado = 4;}

if (P_VS == true){  //Guarda consigna, incrementa o decrementa
  set_lim_vel = set_lim_vel + 1;}
if (P_VB == true){
  set_lim_vel = set_lim_vel - 1;}

if (P_mode_C == true){  //Desconexión de sistemas automaticos
acelerador_out = acelerador_in;
estado = 0;}

break;

  }
```

10.2 Función velocímetro()

```
void velocimetro() {

   // read the analog in value:
  Bobina11 = analogRead(analogInPin0);
  Bobina12 = analogRead(analogInPin1);
  Bobina21 = analogRead(analogInPin2);
  Bobina22 = analogRead(analogInPin3);

   if ( Bobina21 > Bobina22) {
    vel = (-0.0539*Bobina21 + 65);}
      else if ( Bobina22 > Bobina21) {
        if ( Bobina22 > Bobina12) {
              if (Bobina11 > Bobina12) {
                vel = (0.0571*Bobina11 + 130);
              } else {
                vel = (-0.0543*Bobina12 + 142);}
         }
        else {
        vel = ( 0.0557 * Bobina22 + 53);}

  } else {
    vel = 2000;
    }
}
```

10.3 Función mando ()

```
void mando() {

 valor_mando = analogRead(analogInPin6);

if (valor_mando < 100) { // Analiza boton pulsado y antirrebote
errormando = 1;} //Error por mando desconectado
else if (valor_mando < 300) { //Mando conectado
errormando = 0;
cuentaVS = 0;
cuentaVB = 0;
cuentaSS = 0;
cuentaSB = 0;
cuentamode = 0;
P_mode_L = false;
P_mode_C = false;
P_VS = false;
P_VB = false;
P_SS = false;
P_SB = false;}
else if (valor_mando < 570) { // Pulsador mode
  cuentamode = cuentamode+1;}
else if (valor_mando < 700) { //Pulsador Seek subir
  cuentaSS = cuentaSS+1;
  P_SS = false;}
else if (valor_mando < 820) { //Pulsador Seek bajar
  cuentaSB = cuentaSB+1;
  P_SB = false;}
else if (valor_mando < 930) { //Pulsador Velocidad subir
  cuentaVS = cuentaVS+1;
  P_VS = false;}               //Condicion para generar pulsos con
entrada activa
else if (valor_mando < 1020) { //Pulsador Velocidad bajar
  cuentaVB = cuentaVB+1;       //Condicion para generar pulsos con
entrada activa
  P_VB = false;}

if (cuentamode > 200){//Valida pulsacion
P_mode_C = false;
P_mode_L = true;}
if (cuentamode == 10)//Valida pulsacion
P_mode_C = true;
if (cuentaVS > 60){//Valida pulsación
P_VS = true;
cuentaVS = 0;}
if (cuentaVB > 60){//Valida pulsación
P_VB = true;
cuentaVB = 0;}
if (cuentaSS == 200)//Valida pulsación
P_SS = true;
if (cuentaSB == 200)//Valida pulsación
P_SB = true;
}
```

10.4 Función bluetooht ()

```
void bluetooht(){
 if(Serial3.available()>0){           // Si el puerto serie esta
habilitado
        char_blue = Serial3.read();      // Lee lo que llega por el
puerto Serie
 }

 if(char_blue== '1'){                    // on/off de pin 136
      digitalWrite(13,HIGH);
 }
 if(char_blue== '2'){                    // on/off de pin 13
      digitalWrite(13,LOW);
 }

  if(char_blue== 'R'){       // envia todos los datos cuando la
aplicación lo solicita (R)
  Serial3.print(estado);
  Serial3.print("|");
  Serial3.print(acelerador_out);
  Serial3.print("|");
  Serial3.print(vel);
  Serial3.print("|");
  Serial3.print(distancia_radar);
  Serial3.print("|");
  Serial3.print(error,2);
  Serial3.print("|");
  Serial3.print(ITerm,2);
  Serial3.print("|");
  }
 char_blue=0;
 }
```

10.5 Función acelera ()

```
void acelera() {
// read the analog in value:
  acelerador_in = map(analogRead(acel_in_4), 220, 750, 0, 100); //
Conexión a pin 4 del acelerador, escalado %
  // map it to the range of the analog out:
acelerador_in_5 = map(analogRead(acel_in_5), 220, 750, 0, 100); //
Conexión a pin 5 del acelerador, escalado %
acelerador_in_7 = map(analogRead(acel_in_4), 220, 750, 0, 100); //
Conexión a pin 7 del acelerador, escalado %

// Función para la seguridad del correcto funcionamiento, comprueba
que no exista una desviación de 10% entre señales del acelerador.
if ((((acelerador_in + acelerador_in_5 + acelerador_in_7)/3)-
acelerador_in) > 10)
error_acelerador = true;
if ((((acelerador_in + acelerador_in_5 + acelerador_in_7)/3)-
acelerador_in) > 10)
error_acelerador = true;

  outputValue2 = map(acelerador_out, 0, 100, 66, 199);
//65.79=1.29*255/5  pin 4
  outputValue3 = map(acelerador_out, 0, 100,221,43 ); // pin 5
  outputValue4 = map(acelerador_out, 0, 100, 36, 171); // pin 7
  // change the analog out value:
  analogWrite(analogOutPin2, outputValue2);
  analogWrite(analogOutPin3, outputValue3);
  analogWrite(analogOutPin4, outputValue4);
}
```

10.6 Función lazo_control ()

```
void lazo_control()
{
//Función para la ejecución por periodos de muestreo del control
t_control = (ultimo_scan - f_control);
if(t_control > t_muestreo_control)
{
// Calcula el error del lazo de control.
error = Setpoint - vel;

// Calcula el error acumulado
ITerm = ITerm + (ki * error);

//Aunti windup del error acumulado
if (ITerm > 20){
  ITerm = 20; }
if (ITerm < (-20){
  ITerm = 20; }


// Calculamos la función de salida del PI.
Output = kp * error + ITerm;
Outputint = Output; // convierte el decimal a entero
actcontrol = true; //Refistro de activación dela función control. Bit
para las acciones temporizadas (serial PC)

lastInput = Input;
f_control = ultimo_scan;

if (Outputint > 75){ //Aunti windup
Outputint = 75;}
if (Outputint < 4){ //Aunti windup
Outputint = 4;}
// Guardamos el valor de algunas variables para el próximo recálculo.

}
}
```

10.7 Función f_gps ()

```
void f_gps()
{

     // Intentar recibir secuencia durante un segundo
t_gps = (ultimo_scan - tf_gps);
if(t_gps>=t_muestreo_gps)
{
      while (Serial2.available())
      {
         char c = Serial2.read();
         if (gps.encode(c)){ // Nueva secuencia recibida
            newData = true;
            tf_gps = millis();}
      }
   }
   if (newData){
    flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6;
    flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6;
    lista_gps();
    //Calculo de distancia al punto de referencia más próximo dado por
la función lista_gps()
    d_x = R * sin((dif_LAT_ultima)* PI/180);
    d_y = R * sin((dif_LON_ultima)* PI/180);
    distancia_radar = sqrt(d_y*d_y+d_x*d_x);
//

//
    }

     gps.stats(&chars, &sentences, &failed);
//   Serial.print(" CHARS=");
//   Serial.print(chars);
//   Serial.print(" SENTENCES="); // mayor que cero empieza a emviar,
sino incementa no ha actualizado
//   Serial.print(sentences);
//   Serial.print(" CSUM ERR=");
//   Serial.println(failed);

   if (newData)
   {

      gps.f_get_position(&flat, &flon, &age);
      velgps = gps.f_speed_kmph();
      direccion = gps.f_course();
      altitud = gps.f_altitude();

//      Serial.print("LAT=");
//      Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 :
flat, 6);
//      Serial.print(" LON=");
//      Serial.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 :
flon, 6);
//      Serial.print(" SAT=");
//      Serial.print(gps.satellites() ==
TinyGPS::GPS_INVALID_SATELLITES ? 0 : gps.satellites());
//      Serial.print(" PREC=");
```

```
//      Serial.print(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 :
gps.hdop());
//      Serial.print(" Km/h=");
//      Serial.print(velgps);
//      Serial.print(" Direcciion=");
//      Serial.print(direccion);
//      Serial.print(" Altitud=");
//      Serial.print(" Distancia x =");
//      Serial.print(d_x);
//      Serial.print(" Distancia y =");
//      Serial.print(d_y);
//      Serial.print(" Distancia radar =");
//      Serial.print(distancia_radar);

   }


}
```

10.8 Función lista_gps()

```
float dif_LAT_ultima = 0;
float dif_LON_ultima = 0;
int p = 0;

// Radar {Lobete, Sequero, Sequero2, Duques, Universidad}
float ref_LAT[] = { 42.465729, 42.458930, 42.458887, 42.434203,
42.429933};
float ref_LON[] ={ -2.422874, -2.432625, -2.452598, -2.2782757, -
2.277204};
float ref_direccion[]={0, 0, 0, 0, 0}; //Orientación en la que detecta
el radar, 0 para ambos sentidos

void lista_gps()
{

dif_LAT_ultima = flat - ref_LAT[p];//Guarda la diferencia con puntero
ultima
dif_LON_ultima = flon - ref_LON[p];//Guarda la diferencia con puntero
ultima

if((flat - ref_LAT[p++]) < dif_LAT_ultima){
p = p++; //Incrementar el puntero para la coordeana mas proxima
}
if((flon - ref_LAT[p--]) < dif_LAT_ultima){
p = p--; //Decrementar el puntero para la coordeana mas proxima
}

dif_LAT_ultima = flat - ref_LAT[p];//Guarda la diferencia con puntero
actual
dif_LON_ultima = flon - ref_LON[p];//Guarda la diferencia con puntero
actual

}
```

UNIVERSIDAD DE LA RIOJA

PLANOS

Desarrollo de Sistema de Control de Velocidad para automóvil sincronizado con la base de datos de cinemómetros de la DGT

UNIVERSIDAD DE LA RIOJA
GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

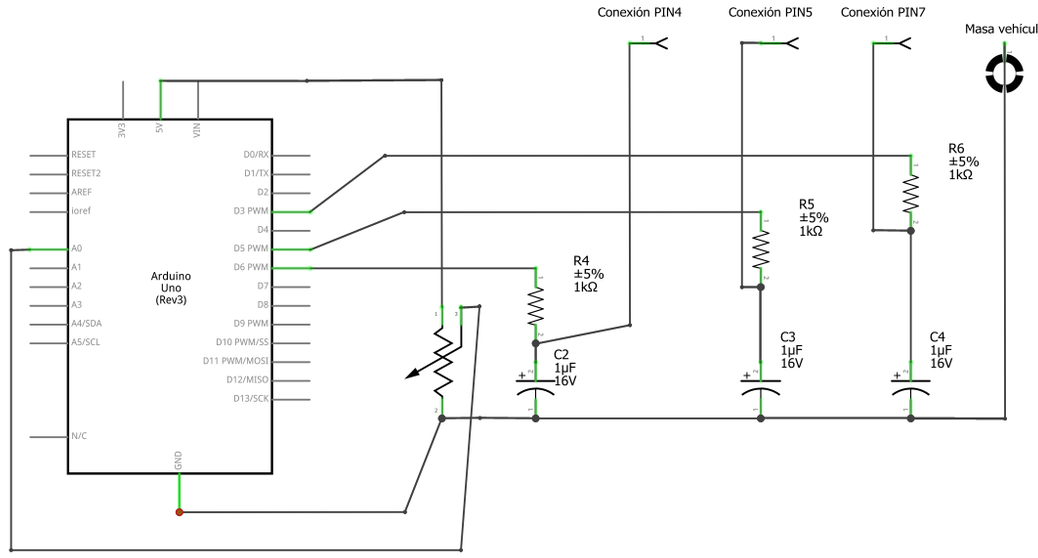
Firma del presente documento:

Iván Peña Moreno

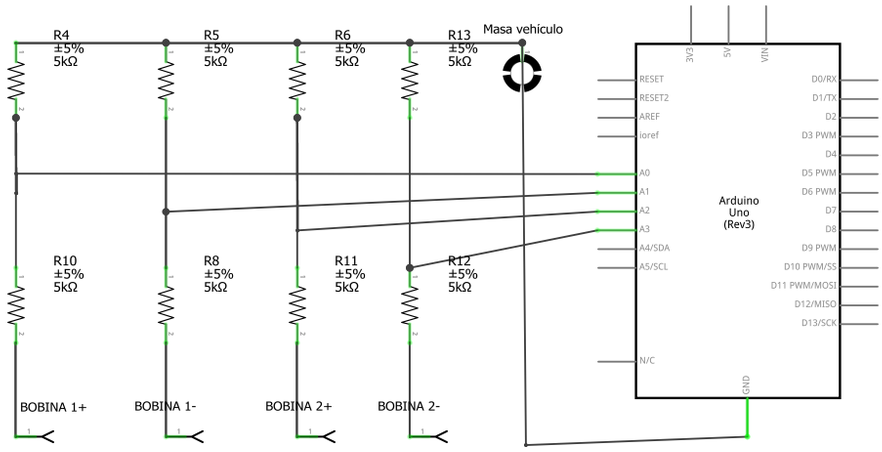
UNIVESIDAD DE LA RIOJA

Logroño, 3 de Marzo de 2019

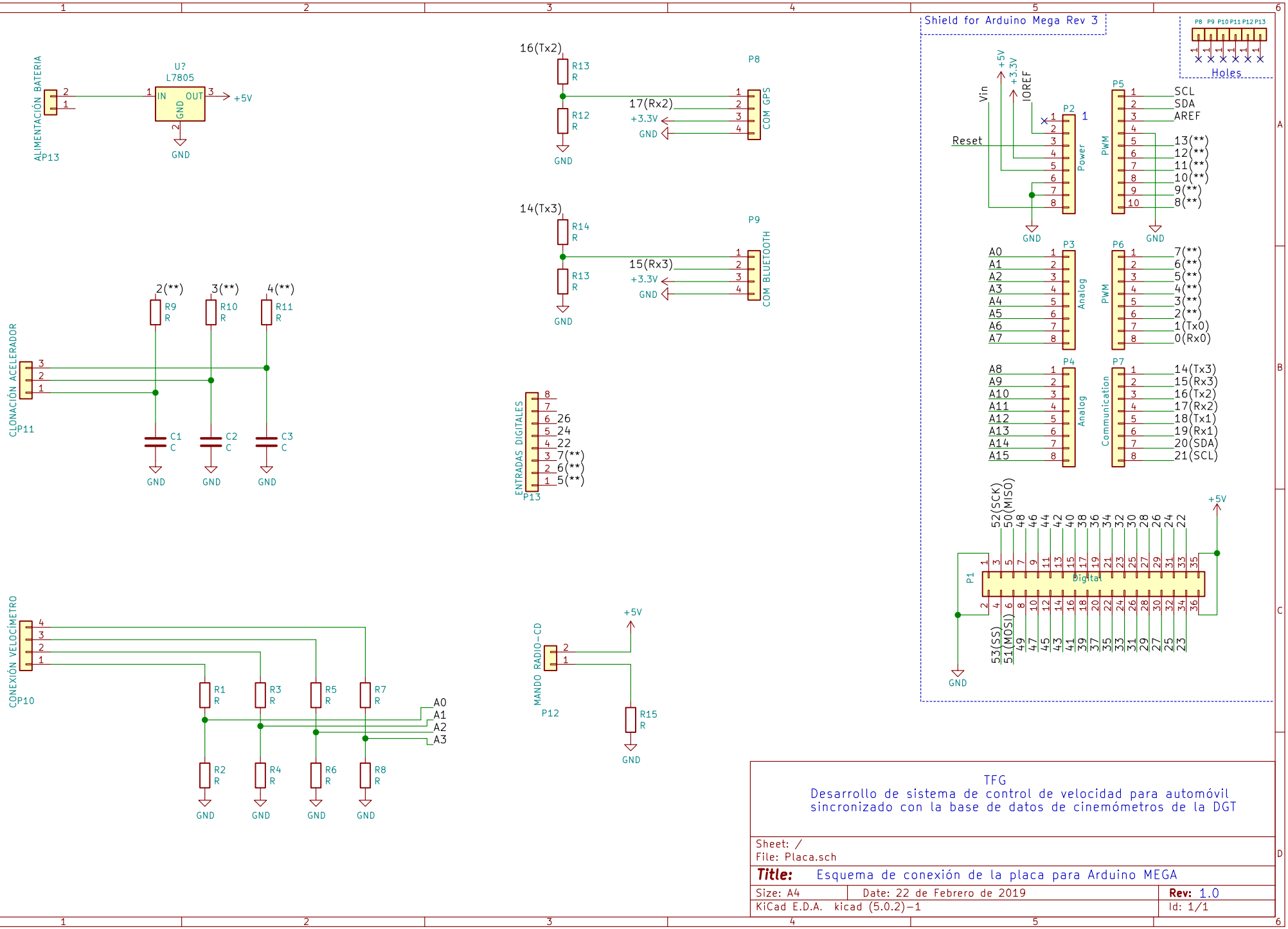
PLANOS



Proyecto	TFG Control de velocidad		
*Filename	Conexíon clonación acelerador	Rev	
Fecha	13 feb 2019 20:02:01	Hoja	1/2



Proyecto	TFG Control de velocidad		
*Filename	Conexión indicador velocidad	Rev	
Fecha	08 ene 2019 20:02:01	Hoja	1/1



TFG

Desarrollo de sistema de control de velocidad para automóvil sincronizado con la base de datos de cinemómetros de la DGT

Sheet: /
File: Placa.sch

Title: Esquema de conexión de la placa para Arduino MEGA

Size: A4	Date: 22 de Febrero de 2019	**Rev:** 1.0
KiCad E.D.A. kicad (5.0.2)-1		Id: 1/1

UNIVERSIDAD DE LA RIOJA

PLIEGO DE CONDICIONES

Desarrollo de Sistema de Control de Velocidad para automóvil sincronizado con la base de datos de cinemómetros de la DGT

UNIVERSIDAD DE LA RIOJA
GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Firma del presente documento:

Iván Peña Moreno

UNIVESIDAD DE LA RIOJA

Logroño, 3 de Marzo de 2019

PLIEGO DE CONDICIONES

PLIEGO DE CONDICIONES

1. Disposiciones generales

1.1 Disposiciones de carácter general

1.1.1 Objeto

La finalidad de este Pliego es la de fijar los criterios de la relación que se establece entre los agentes que intervienen en los trabajos y acciones definidos en el presente proyecto y servir de base para la realización del mismo. Así como la ordenación de las condiciones técnicas, económicas y legales que han de regir en su ejecución.

1.1.2 Contrato del proyecto

Se recomienda la contratación de la ejecución de los trabajos y acciones pertinentes con arreglo a los documentos del proyecto y en cifras fijas. A tal fin, se ofrece la documentación necesaria para la realización del contrato del proyecto.

1.1.3 Documentación del proyecto

Integran el proyecto los siguientes documentos, relacionados por orden de prelación atendiendo al valor de sus especificaciones, en el caso de posibles interpretaciones, omisiones o contradicciones:

- Planos generales y de detalles
- El presente Pliego de Condiciones
- Memoria
- Anexos
- Mediciones y Presupuestos

En el caso de posibles variaciones en la interpretación, prevalecen las especificaciones literales sobre las gráficas y las cotas, si procede, sobre las medidas a escala tomadas de los planos.

2. Disposiciones relativas a materiales y trabajos

Se describen las disposiciones básicas a considerar en la ejecución material del proyecto, relativas a los materiales y trabajos, así como a la recepción del presente proyecto.

2.1 Inicio del proyecto y ritmo de ejecución

La ejecución de los trabajos comenzará en el plazo especificado en el respectivo contrato, desarrollándose de manera adecuada para que dentro de los períodos parciales señalados serealicen los trabajos, de modo que la ejecución total se lleve a cabo dentro del plazo establecido en el contrato.

2.2 Orden de los trabajos

La determinación del orden de los trabajos es, generalmente, facultad del Ingeniero director del proyecto, salvo en aquellos casos en que, por circunstancias de naturaleza técnica, se estime conveniente su variación por parte del mismo.

2.3 Interpretaciones, aclaraciones y modificaciones del proyecto

El responsable de la fabricación y montaje podrá requerir del Ingeniero director del proyecto las instrucciones o aclaraciones que se precisen para la correcta interpretación y ejecución de los trabajos proyectados.

Cuando se trate de interpretar, aclarar o modificar preceptos del Pliego de Condiciones o indicaciones de los planos, croquis, órdenes e instrucciones correspondientes, se comunicarán necesariamente por escrito al responsable de la fabricación y montaje, estando éste a su vez obligado a devolver los originales o las copias, suscribiendo con su firma el enterado, que figurará al pie de todas las órdenes, avisos e instrucciones que reciba.

2.4 Materiales, aparatos y equipos defectuosos

Cuando los materiales, aparatos, equipos y elementos de instalaciones no fuesen de la calidad y características técnicas prescritas en el proyecto, no tuvieran la preparación en él exigida o cuando, a falta de prescripciones formales, se reconociera o demostrara que no son los adecuados para su fin, se dará la orden al

responsable de la fabricación y desarrollode sustituirlos por otros que satisfagan las condiciones o sean los adecuados al fin al que se destinen.

En el caso de que los materiales, aparatos, equipos o elementos de instalaciones fueran defectuosos, pero aceptables a juicio del responsable de la fabricación y desarrollo, se recibirán con la rebaja del precio que aquél determine, a no ser que el responsable prefiera sustituirlos por otros en condiciones.

2.5 Procedencia de materiales, aparatos y equipos

El responsable de la fabricación tiene libertad de proveerse de los materiales, aparatos y equipos de todas clases donde considere oportuno y conveniente para sus intereses, excepto en aquellos casos en los se preceptúe una procedencia y características específicas en el proyecto. Obligatoriamente, y antes de proceder a su empleo, acopio y puesta en servicio, el responsable de la fabricación deberá presentar una lista completa de los materiales, aparatos y equipos que vaya a utilizar, en la que se especifiquen todas las indicaciones sobre sus características técnicas, marcas, calidades, procedencia e idoneidad de cada uno de ellos.

3. Disposiciones económicas

Las condiciones económicas fijan el marco de relaciones económicas para el abono y recepción del proyecto. Tienen un carácter subsidiario respecto al contrato del proyecto, establecido entre las partes que intervienen.

3.1 Contrato de ejecución

Se aconseja que se firme el contrato de ejecución, entre el destinatario y el responsable de la fabricación, desarrollo y montaje, antes de iniciarse los trabajos y/o acciones necesarios para la realización material del proyecto.

Al director del proyecto se le facilitará una copia del contrato de ejecución, para poder certificar en los términos pactados. El contrato de ejecución deberá prever las posibles interpretaciones y discrepancias que pudieran surgir entre las partes, por lo que es conveniente que se especifiquen y determinen con claridad, como mínimo, los siguientes puntos:

- Documentos a aportar por el responsable de fabricación y desarrollo.

- Condiciones de inicio de los trabajos.
- Responsabilidades y obligaciones del responsable de fabricación: Legislación laboral.
- Presupuesto del responsable de fabricación.
- Forma de pago: Certificaciones.
- Retenciones en concepto de garantía (nunca menos del 5%).
- Plazos de ejecución: Planning.
- Retraso del proyecto
- Penalizaciones.

3.2 Precios

El objetivo principal de la elaboración del presupuesto es anticipar el coste del proceso de construir el proyecto. Se descompondrá el presupuesto en unidades, componente menor que se contrata y certifica por separado, y basándose en esos precios, se calculará el presupuesto.

3.2.1 Precio básico

Es el precio por unidad (ud., m, kg, etc.) de un material, (incluido su transporte, descarga, embalajes, etc.) o el precio por hora de la maquinaria y de la mano de obra.

3.2.2 Precios contradictorios

Sólo se producirán precios contradictorios cuando el Ingeniero director del Proyecto decida introducir unidades o cambios de calidad en alguna de las previstas, o cuando sea necesario afrontar alguna circunstancia imprevista. El responsable de la fabricación, desarrollo y montaje siempre estará obligado a efectuar los cambios indicados.

A falta de acuerdo, el precio se resolverá contradictoriamente entre el Director del Proyecto y el responsable de la fabricación antes de comenzar la ejecución de los trabajos y en el plazo que determine el contrato de ejecución o, en su defecto, antes de quince días hábiles desde que se le comunique fehacientemente al Director del Proyecto.

Si subsiste la diferencia, se acudirá, en primer lugar, al concepto más análogo dentro del cuadro de precios del proyecto y, en segundo lugar, al banco de precios de uso más frecuente en la localidad. Los contradictorios que hubiese se referirán siempre a los precios unitarios de la fecha del contrato de ejecución.

4. Preinscripciones técnicas particulares

Para facilitar la labor a realizar, por parte del Director del Proyecto, para el control de recepción de los productos, equipos y sistemas que se suministren en el presente proyecto, se especifican las características técnicas que deberán cumplir los productos, equipos y sistemas suministrados.

Los productos, equipos y sistemas suministrados deberán cumplir las condiciones que sobre ellos se especifican en los distintos documentos que componen el Proyecto. Asimismo, sus calidades serán acordes con las distintas normas que sobre ellos estén publicadas y que tendrán un carácter de complementariedad a este apartado del Pliego.

Tendrán preferencia en cuanto a su aceptabilidad aquellos materiales que estén en posesión de Documento de Idoneidad Técnica que avale sus cualidades, emitido por Organismos Técnicos reconocidos.

Este control de recepción de productos, equipos y sistemas comprenderá:

- El control de la documentación de los suministros

- El control mediante distintivos de calidad o evaluaciones técnicas de idoneidad

- El control mediante ensayos por parte del responsable de la fabricación debe existir obligación de comunicar a los suministradores de productos las cualidades que se exigen para los distintos materiales, aconsejándose que previamente al empleo de los mismos se solicite la aprobación del Director del Proyecto y de las entidades y laboratorios encargados del control de calidad.

El responsable de la fabricación del chasis será responsable de que los materiales empleados cumplan con las condiciones exigidas, independientemente del nivel de control de calidad que se establezca para la aceptación de los mismos.

El responsable de la fabricación notificará al Director del Proyecto, con suficiente antelación, la procedencia de los materiales que se proponga utilizar, aportando, cuando así lo solicite el Director del Proyecto, las muestras y datos necesarios para decidir acerca de su aceptación.

Estos materiales serán reconocidos por el Director del Proyecto antes de su empleo, sin cuya aprobación no podrán ser acopiados ni se podrá proceder a su utilización.

Asimismo, aún después de colocados, aquellos materiales que presenten defectos no percibidos en el primer reconocimiento, siempre que vaya en perjuicio del buen acabado del proyecto, serán retirados. Todos los gastos que ello ocasionase

serán a cargo del responsable de la fabricación. El hecho de que el responsable subcontrate cualquier partida no le exime de su responsabilidad.

5. Materiales y componentes

- Arudino UNO. Es una placa de desarrollo con un microcontrolador. Esta placa incorpora un microcontrolador fabricado por ATMEL.

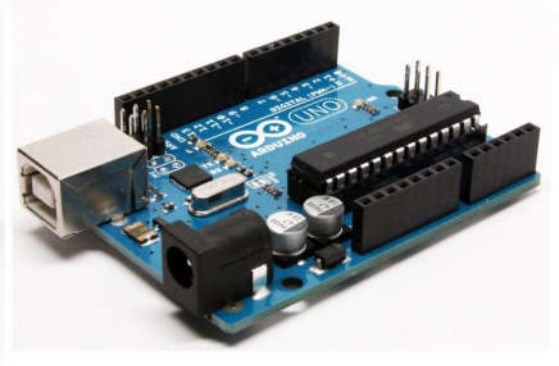


Ilustración 1: Placa de desarrollo Arduino UNO

Sus características son las siguientes:

- Voltaje: 5V
- Voltaje de entrada (recomendado): 7-12V
- Voltaje de entrada (límites): 6-20V
- Pines de entradas y salidas digitales: 14
- Pines de entrada analógica: 6
- Corriente en pines de entrada-salida: 40mA
- Corriente en pin de 3.3V: 50mA
- Memoria Flash: 32KB
- SRAM: 2KB
- EEPROM: 1KB
- Velocidad reloj: 16MHz
- Dimensiones: 68.6x53.4mm

- Arduino MEGA. Es una placa de desarrollo con microprocesador de ATMEL ATmega2560.

Ilustración 2: Placa de desarrollo Arduino MEGA

- **Microcontrolador:** ATmega2560
- **Voltaje Operativo:** 5V
- **Voltaje de Entrada:** 7-12V
- **Voltaje de Entrada(límites)**: 6-20V
- **Pines digitales de Entrada/Salida:** 54 (de los cuales 15 proveen salida PWM)
- **Pines análogos de entrada:** 16
- **Corriente DC por cada Pin Entrada/Salida:** 40 mA
- **Corriente DC entregada en el Pin 3.3V:** 50 mA
- **Memoria Flash:** 256 KB (8KB usados por el bootloader)
- **SRAM:** 8KB
- **EEPROM:** 4KB
- **ClockSpeed:** 16 MHz

- Circuito integrado LM2917N. Un integrado que da la posibilidad de conversión de pequeñas señales de sensores magnéticos a señales analógicas lineales amplificadas.
  Este integrado en su circuito interno incluye: un comparador de tensión en la entrada con una función de histéresis, una bomba de carga como convertidor frecuencia en tensión y un amplificador operacional con un transistor de salida.
  El diagrama en bloques del circuito interno debe corresonder a el siguiente diagrama:

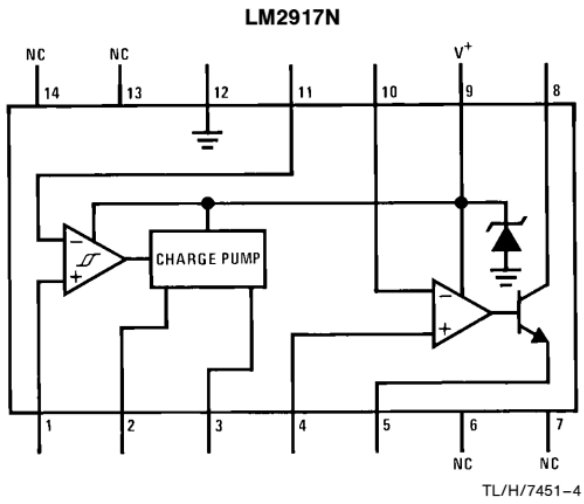


Ilustración 3: Diagrama de bloque de LM2917N

- Interfaz CAN. Placa de desarrollo para la gestión de Bus CAN, con conexión de comunicación SPI.

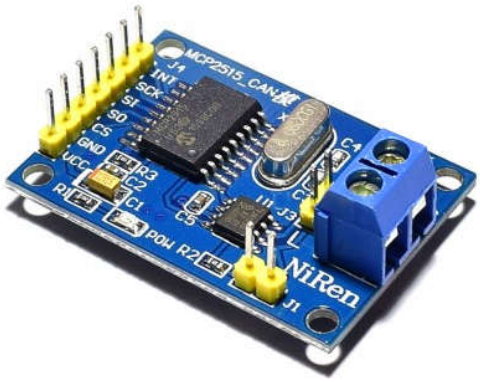


Ilustración 4: Interfaz CAN MCP 2515

Características:

- MCP2515: Controlador autónomo CAN. Compatible con CAN 2.0A y CAN 2.0B, por lo que puede trabajar con tramas extendidas y con tramas estándar.
- Interface SPI de hasta 10 MHz
- MCP2551: Es un transceiver CAN de alta velocidad (hasta 1MB/s) y hará de puente entre el controlador MCP2515 y el propio bus físico CAN. Implementa compatibilidad con el estándar ISO-11898.
- Data Frames standard (11 bit) y extendidos (29 bit)
- 2 buffers de recepción con priorización de mensaje.
- 2 LEDs indicadores.
- Cristal oscilador de 8 MHz: Es el encargado de sincronizar el controlador con el bus.

- Modulo GPS con integrado NEO-6M. Módulo para la geolocalización con una pequeña antena con baja precisión incluida.

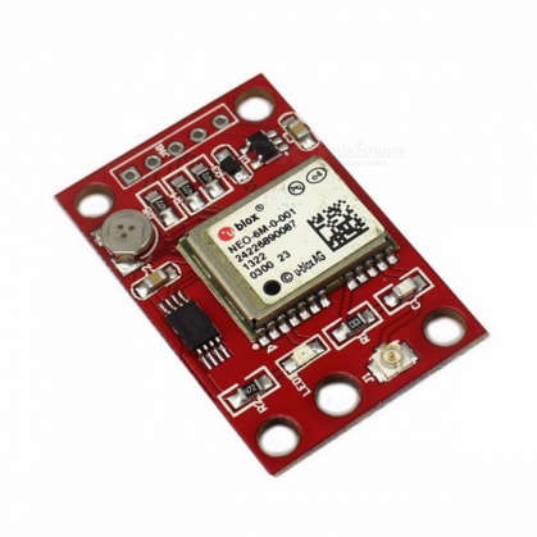


Ilustración 5: Módulo NEO-6M

Este módulo funcionara alimentado con una tensión de alimentación de 3 a 5V y se comunicará a través de puerto serie. La comunicación funcionará por defecto a 9600bps y transmitirá la información a través del estándar NMEA.

Además de los pines de conexión y comunicación, el modulo dispondrá de dos leds indicadores de información, uno de alimentación correcta y otro de señal de satélites.

Características:

- Modulo GPS Ublox NEO-6M
- Comunicación serial
- Voltaje de alimentación: (3.5 – 5)VDC
- Antena cerámica activa incluida
- LED indicador de señal
- Tamaño de antena 22x22mm
- Tamaño de modulo 23x30mm
- Batería incluida
- BAUDRATE: 9600
- EEPROM para guardar configuración de parámetros
- Sistema de coordenadas: WGS-84
- Sensibilidad de captura -148dBm
- Sensibilidad de rastreo: -161 dBm
- Máxima altura medible: 18000

- Máxima velocidad 515 m/s
- Exactitud: 1micro segundo
- Frecuencia receptora: L1 (1575.42 Mhz)
- Código C/A 1.023 Mhz
- Tiempo de inicio primera vez: 38s en promedio
- Tiempo de inicio : 35s en promedio

➢ Módulo Bluetooth. Módulo para la comunicación Bluetooth entre un microcontrolador y un Smartphone. Deberá tener comunicación serie para la comunicación con el microcontrolador.

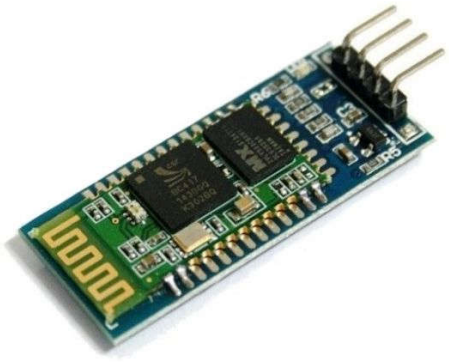


Ilustración 6: Módulo HC-06

Este modulo que se ve en la ilustración llevará integrado un regulador de tensión admitiendo así tensiones desde los 3 a los 6V de alimentación y los pines respectivos para la comunicación serie con el microcontrolador, haciendo así de puente entre el Smartphone y el Arduino.

Características:

- Especificación bluetooth v2.0 + EDR (Enhanced Data Rate)
- Módulo montado en tarjeta con regulador de voltaje y 4 pines suministrando acceso a VCC, GND, TXD, y RXD
- Consumo de corriente: 30 mA a 40 mA
- Voltaje de alimentación: 3,3 V a 6 V
- Voltaje de operación: 3.3V
- Dimensiones totales: 1.7 cm x 4 cm aprox.
- Temperatura de operación: -25 ºC a +75 ºC
- Modo esclavo (Solo puede operar en este modo)
- Puede configurarse mediante comandos AT (Deben escribirse en mayúscula)

- Chip de radio: CSR BC417143
- Frecuencia: 2.4 GHz, banda ISM
- Modulación: GFSK (Gaussian Frequency Shift Keying)
- Antena de PCB incorporada
- Potencia de emisión: ≤ 6 dBm, Clase 2
- Alcance 5 m a 10 m
- Sensibilidad: ≤ -80 dBm a 0.1% BER
- Velocidad: Asincrónica: 2 Mbps (max.)/160 kbps, sincrónica: 1 Mbps/1 Mbps
- Seguridad: Autenticación y encriptación (Password por defecto: 1234)
- Perfiles: Puerto serial Bluetooth

➢ Conversor digital analógico. El MCP4725 es un conversor controlado a través de I2C de sencilla configuración y control a través del microcontrolador, precisión de 12 bits y muy económico.

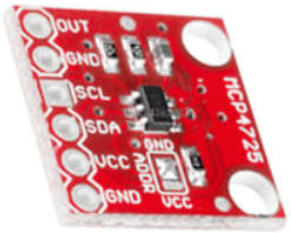


Ilustración 7: MCP4725

➢ Otros componentes electrónicos.

- 8 resistencias estándar de 5.2kΩ.
- 3 resistencias de 10kΩ.
- 3 condensadores de 1 µF.
- Placa pcb virgen para la impresión. Dimensiones: 18mm×10mm.
- 29 metros de cable. Sección: 0,25mm^2 .
- Regulador de tensión lineal 7805 1A

- 20 conectores 2.54 macho para la instalación en pcb.
- 20 conectores 2.54 hembra para la instalación en pcb.

Las características de los materiales suministrados deben estar documentadas de forma que puedan compararse con los requisitos preestablecidos. Además, los materiales deben poderse identificar en todas las etapas de fabricación, de forma única y por un sistema apropiado.

La identificación puede basarse en registros documentados para lotes de producto asignados a un proceso común de producción, pero cada componente debe tener una marca duradera, distinguible, que no le produzca daño y resulte visible tras el montaje.

5.1 Manipulación y almacenamiento de los materiales

El material y los componentes deben almacenarse siguiendo las instrucciones de su fabricante y no usarse si ha superado la vida útil en almacén especificada. Si por la forma o el tiempo de almacenaje pudieran haber sufrido un deterioro importante, antes de su utilización deben comprobarse que siguen cumpliendo con los requisitos establecidos.

Los componentes deben manipularse y almacenarse de forma segura, evitando que se produzcan daños permanentes y de manera que los daños superficiales sean mínimos.

UNIVERSIDAD DE LA RIOJA

PRESUPUESTO

Desarrollo de Sistema de Control de Velocidad para automóvil sincronizado con la base de datos de cinemómetros de la DGT

UNIVERSIDAD DE LA RIOJA
GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Firma del presente documento:

Iván Peña Moreno

UNIVESIDAD DE LA RIOJA

Logroño, 3 de Marzo de 2019

PRESUPUESTO

1.-Listado de precios unitarios

Código	Ud	Descripción	Precio (€)
CDC001	h	Montaje del sistema por instalador autorizado	59.10
CDC002	h	Ingeniero electrónico	43.60
CDC003	u	Arduino Uno	23.45
CDC004	u	Protoboard	6.51
CDC005	u	Módulo MCP2515/2551	8.24
CDC006	u	Arduino MEGA	35.25
CDC007	u	NEO-6M GPS	17.14
CDC008	u	Cables 0.25mm	0.12
CDC009	u	Módulo Bluetooth	12.25
CDC010	u	Placa baquelita v303 100x200	3.55
CDC011	u	Resistencias 3.7K, 10K	0.11
CDC012	u	Separador hexagonal métrica 3 x 5mm	0.14
CDC013	u	Condendadores 1uF	0.40
CDC014	u	Conectores 2.54 hembra/macho	0.18

2.-Listado de precios descompuestos

2.1 Capítulo I: Estudio de la solución

Código	Cantidad (Uds)	Descripción	Precio (€)	total (€)
CDC007	1 u	NEO-6M GPS	17.14	17.14
CDC006	1u	Arduino MEGA	35.25	35.25
CDC003	1 u	Arduino Uno	23.45	23.45
CDC004	1 u	Protoboard	6.51	6.51
CDC005	1 u	Módulo MCP2515/2551	8.24	8.24
CDC013	3u	Condendadores 1uF	0.40	1.2
CDC008	10 u	Cables 0.25MM	0.12	1.2
CDC011	2u	Resistencias 3.7K	0.11	0.22
CDC011	12u	Resistencias 10K	0.11	1.1
CDC002	120 h	Ingeniero electrónico	44	2025

Suma la partida..... 5373.31€

Beneficio industrial (7%)..... 376.13€

Total partida.....5749.44€

2.2 Capítulo II: Desarrollo de prototipo

Código	Cantidad (Uds)	Descripción	Precio (€)	Total (€)
CDC010	u	Placa baquelita v303 100x200	3.55	3.55
CDC002	6 h	Ingeniero electrónico (diseño electónico)	44	264
CDC002	16 h	Ingeniero electrónico (programación Aplicación móvil)	44	704
CDC002	20 h	Ingeniero electrónico (programación Aplicación móvil)	44	880
CDC001	10h	Montaje del sistema por instalador autorizado	59.10	354.6

Suma la partida..... 2206.15€

Beneficio industrial (7%)..... 154.43€

Total partida..... 2360.58€

3.-Resumen del presupuesto del proyecto

Capítulo	Resumen	Importe (€)
Capítulo I	Estudio de la solución	5749.44
Capítulo II	Desarrollo del prototipo	2360.58

Total ejecución.....8110.02€

IVA 21%.....1703.1€

TOTAL PRESUPUESTO CONTRATO.....9813.12€

El precio total del proyecto asciende a la cantidad de NUEVE MIL OCHOCIENOS TRECE EUROS CON DOCE CÉNTIMOS (9813.12€).

En Logroño, a 4 de marzo de 2019

Fdo: Iván Peña Moreno